24th International Conference on Database Theory

ICDT 2021, March 23–26, 2021, Nicosia, Cyprus

Edited by

Ke Yi

Zhewei Wei

Editors

Ke Yi
The Hong Kong University of Science and Technology, Hong Kong
yike@ust.hk

Zhewei Wei
Renmin University of China, China
zhewei@ruc.edu.cn

ACM Classification 2012
Information systems → Data management systems; Information systems → Database design and models; Information systems → Database query processing; Information systems → Query languages; Information systems → Relational database model; Information systems → Parallel and distributed DBMSs; Information systems → Information integration; Information systems → Stream management; Theory of computation → Incomplete, inconsistent, and uncertain databases; Theory of computation → Complexity theory and logic; Theory of computation → Database theory

ISBN 978-3-95977-179-5

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-179-5.

Publication date
March, 2021

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ICDT.2021.0

ISBN 978-3-95977-179-5 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

24th International Conference on Database Theory (ICDT 2021).
Editors: Ke Yi and Zhewei Wei
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The 24. International Conference on Database Theory (ICDT 2021) was held in Nicosia, Cyprus, from March 23 to 26, 2021. The Program Committee has selected 20 research papers out of 42 submissions for publication at the conference. It has further decided to give the Best Paper Award to *Answer Counting Under Guarded TGDs* by Cristina Feier, Carsten Lutz, and Marcin Przybyłko. We congratulate the winners! Apart from the 20 regular papers, these proceedings include abstracts for the invited (shared) EDBT/ICDT keynotes by Pablo Barceló (Pontificia Universidad Católica de Chile) and by Julia Stoyanovich (New York University).

A committee formed by Yael Amsterdamer, Rasmus Pagh, and Pierre Senellart has decided to give the Test of Time Award for ICDT 2021 to the ICDT 2011 paper *Knowledge compilation meets database theory: compiling queries to decision diagrams* by Abhay Jha and Dan Suciu. We congratulate also the winners of this award!

We would like to thank all people who contributed to the success of ICDT 2021, including the authors of all submitted papers, keynote and invited talk speakers, and, of course, all members of the Program Committee as well as the external reviewers, for the very substantial work that they have invested over the two submission cycles of ICDT 2021. Their commitment and sagacity were crucial to ensure that the final program of the conference satisfies the highest standards. We would also like to thank the ICDT Council members for their support on a wide variety of matters, the local organizers of the EDBT/ICDT 2021 conference, led by General Chairs Demetris Zeinalipour and Panos K. Chrysanthis, for the great job they did in organizing the conference and co-located events. Finally, we wish to acknowledge Dagstuhl Publishing for their support with the publication of the proceedings in the LIPIcs (Leibniz International Proceedings in Informatics) series.

Ke Yi and Zhewei Wei
March 2021

24th International Conference on Database Theory (ICDT 2021).
Editors: Ke Yi and Zhewei Wei
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

General Chairs

Demetris Zeinalipour (University of Cyprus)

Panos K. Chrysanthis (University of Cyprus and University of Pittsburgh)

Program Chair

Ke Yi (The Hong Kong University of Science and Technology)

Proceedings Chair

Zhewei Wei (Renmin University of China)

Program Committee

Yael Amsterdamer (Bar Ilan University)

Meghyn Bienvenu (CNRS, University of Bordeaux)

Vladimir Braverman (Johns Hopkins University)

Marco Calautti (University of Trento)

Hubie Chen (Birkbeck, University of London)

Sara Cohen (The Hebrew University)

Martin Grohe (RWTH Aachen University)

Benny Kimelfeld (Technion, Israel Institute of Technology)

Paraschos Koutris (University of Wisconsin-Madison)

Domenico Lembo (Sapienza University of Rome)

Stefan Mengel (CNRS, CRIL)

Matthias Niewerth (University of Bayreuth)

Dan Olteanu (University of Oxford)

Rasmus Pagh (IT University of Copenhagen)

Sudeepa Roy (Duke University)

Atri Rudra (University at Buffalo, SUNY)

Francesco Scarcello (DIMES, University of Calabria)

Srikanta Tirthapura (Apple Inc., Iowa State University)

Stijn Vansummeren (Université Libre de Bruxelles)

Jef Wijsen (University of Mons)

24th International Conference on Database Theory (ICDT 2021).
Editors: Ke Yi and Zhewei Wei

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Antoine Amarill

Mohammad Javad Amiri

Alexandr Andoni

Marcelo Arenas

Anton Belyy

Andrea Calí

Nofar Carmeli

Shaleen Deep

Cibele Freire

Dominik D. Freydenberger

Filippo Furfaro

Gianluigi Greco

Montserrat Hermo

Xiao Hu

Raj Jayaram

Zhengjie Miao

Cristian Molinaro

Frank Neven

Milos Nikolic

Francesco Parisi

Tina Popp

Andrea Pugliese

Juan L. Reutter

Cristian Riveros

Domenico Saccà

Uri Stemmer

Philip Wellnitz

Samson Zhou

24th International Conference on Database Theory (ICDT 2021).
Editors: Ke Yi and Zhewei Wei
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Contributing Authors

Heba Aamer
Kaleb Alway
Antoine Amarilli
Eric Blais
Pierre Bourhis
Noa Bratman
Jan Van den Bussche
Nofar Carmeli
Katrin Casel
Balder ten Cate
Victor Dalmau
Shaleen Deep
Johannes Doleschal
Cristina Feier
Alejandro Grez
Martin Grohe
Xiao Hu
Louis Jachiet
Benny Kimelfeld
Paraschos Koutris
Ester Livshits
Shangqi Lu
Carsten Lutz
Wim Martens
Samuel McCauley
Andrew McGregor
Alexandra Meliou
Zafeiria Moumoulidou
Jeffrey Naughton
Rasmus Pagh
Liat Peterfreund
Marcin Przybyłko
Cristian Riveros
Semih Salihoglu
Markus L. Schmid
Nicole Schweikardt
Nina Mesing Stausholm
Bruhathi Sundarmurthy
Yufei Tao
David Tench
Muhammad Tibi
Hoa T. Vu

24th International Conference on Database Theory (ICDT 2021).
Editors: Ke Yi and Zhewei Wei
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ICDT 2021 Test of Time Award

In 2013, the International Conference on Database Theory (ICDT) began awarding the ICDT test-of-time (ToT) award, with the goal of recognizing one paper, or a small number of papers, presented at ICDT a decade earlier that have best met the “test of time". In 2021, the award recognizes a paper from the ICDT 2011 proceedings that has had the most impact in terms of research, methodology, conceptual contribution, or transfer to practice over the past decade. The award is to be presented during the EDBT/ICDT 2021 Joint Conference, March 23–26, 2021 in Nicosia, Cyprus.

The ICDT 2021 Test of Time Award committee consists of Yael Amsterdamer (Chair), Rasmus Pagh, and Pierre Senellart. After careful consideration and soliciting external assessments, the committee has chosen the following recipient of the 2021 ICDT Test of Time Award:

Knowledge compilation meets database theory: compiling queries to decision diagrams
Abhay Jha and Dan Suciu

There are two main approaches to computing the probability of a query result over probabilistic databases: the extensional approach exploits the structure of the query for efficient evaluation for some classes of queries; the intensional approach first tractably computes a representation of the lineage of the query and then attempts to compute the probability of this Boolean function. This paper shows that a number of cases known to be tractable in the extensional method lead to tractablity in the intensional method because lineages can be produced in specific tractable formalisms (such as OBDDs, FBDDs, d-DNNFs) which are well-studied target compilation classes in knowledge compilation, and for which weighted model counting is tractable. The paper leaves open the major question of whether all tractable cases can be explained in the same manner.

With their work, Jha and Suciu established a strong connection between the fields of knowledge compilation and probabilistic databases, which was both foundational and entirely original. This has sparked research in and across different areas: in database theory in the form of further refinements of the results and progress towards the resolution of the question left open; in database systems by demonstrating that the intensional approach and the use of knowledge compilation techniques are viable for probabilistic query evaluation; and in knowledge compilation by further motivating and reviving interest for the study of the weighted variant of the model counting problem.

Yael Amsterdamer
Bar-Ilan University

Rasmus Pagh
University of Copenhagen

Pierre Senellart
ENS, PSL University

The ICDT Test-of-Time Award Committee for 2021

24th International Conference on Database Theory (ICDT 2021).
Editors: Ke Yi and Zhewei Wei
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Explainability Queries for ML Models and its Connections with Data Management Problems

Pablo Barceló ✉
Universidad Católica de Chile, Macul, Chile

Abstract

In this talk I will present two recent examples of my research on explainability problems over machine learning (ML) models. In rough terms, these explainability problems deal with specific queries one poses over a ML model in order to obtain meaningful justifications for their results. Both of the examples I will present deal with "local" and "post-hoc" explainability queries. Here "local" means that we intend to explain the output of the ML model for a particular input, while "post-hoc" refers to the fact that the explanation is obtained after the model is trained. In the process I will also establish connections with problems studied in data management. This with the intention of suggesting new possibilities for cross-fertilization between the area and ML.

The first example I will present refers to computing explanations with scores based on Shapley values, in particular with the recently proposed, and already influential, SHAP-score. This score provides a measure of how different features in the input contribute to the output of the ML model. We provide a detailed analysis of the complexity of this problem for different classes of Boolean circuits. In particular, we show that the problem of computing SHAP-scores is tractable as long as the circuit is deterministic and decomposable, but becomes computationally hard if any of these restrictions is lifted. The tractability part of this result provides a generalization of a recent result stating that, for Boolean hierarchical conjunctive queries, the Shapley-value of the contribution of a tuple in the database to the final result can be computed in polynomial time.

The second example I will present refers to the comparison of different ML models in terms of important families of (local and post-hoc) explainability queries. For the models, I will consider multilayer perceptrons and binary decision diagrams. The main object of study will be the computational complexity of the aforementioned queries over such models. The obtained results will show an interesting theoretical counterpart to wisdom's claims on interpretability. This work also suggests the need for developing query languages that support the process of retrieving explanations from ML models, and also for obtaining general tractability results for such languages over specific classes of models.

2012 ACM Subject Classification Theory of computation → Models of learning

Keywords and phrases ML models, Explainability, Shapley values, decision trees, OBDDs, deterministic and decomposable Boolean circuits

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.1

Category Invited Talk

Short Bio. Pablo Barceló is a Full Professor at Pontificia Universidad Católica de Chile, where he also acts as Director of the Institute for Mathematical and Computational Engineering. He is the author of more than 80 technical papers, has chaired ICDT 2019, will be chairing ACM PODS 2022, and is currently a member of the editorial committee of Logical Methods in Computer Science. From 2011 to 2014 he was the editor of the database theory column of SIGMOD Record. His areas of interest are database theory, logic in computer science, and the emerging relationship between these areas and machine learning.

Comparing Apples and Oranges: Fairness and Diversity in Ranking

Julia Stoyanovich ✉
New York University, NY, USA

Abstract

Algorithmic rankers take a collection of candidates as input and produce a ranking (permutation) of the candidates as output. The simplest kind of ranker is *score-based*; it computes a score of each candidate independently and returns the candidates in score order. Another common kind of ranker is *learning-to-rank*, where supervised learning is used to predict the ranking of unseen candidates. For both kinds of rankers, we may output the entire permutation or only the highest scoring k candidates, the top-k. Set selection is a special case of ranking that ignores the relative order among the top-k.

In the past few years, there has been much work on incorporating fairness and diversity requirements into algorithmic rankers, with contributions coming from the data management, algorithms, information retrieval, and recommender systems communities. In my talk I will offer a broad perspective that connects formalizations and algorithmic approaches across subfields, grounding them in a common narrative around the value frameworks that motivate specific fairness- and diversity-enhancing interventions. I will discuss some recent and ongoing work, and will outline future research directions where the data management community is well-positioned to make lasting impact, especially if we attack these problems with our rich theory-meets-systems toolkit.

2012 ACM Subject Classification Information systems → Data management systems; Theory of computation → Theory and algorithms for application domains; Social and professional topics → Computing / technology policy

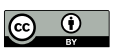
Keywords and phrases fairness, diversity, ranking, set selection, responsible data management

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.2

Category Invited Talk

Funding This work was supported in part by National Science Foundation (NSF) awards No. 1926250, 1934464, 1916505, and 1922658.

Short Bio. Julia Stoyanovich is an Assistant Professor in the Department of Computer Science and Engineering at the Tandon School of Engineering, and the Center for Data Science. She is a recipient of an NSF CAREER award and of an NSF/CRA CI Fellowship. Julia's research focuses on responsible data management and analysis practices: on operationalizing fairness, diversity, transparency, and data protection in all stages of the data acquisition and processing lifecycle. She established the Data, Responsibly consortium, and serves on the New York City Automated Decision Systems Task Force (by appointment by Mayor de Blasio). In addition to data ethics, Julia works on management and analysis of preference data, and on querying large evolving graphs. She holds M.S. and Ph.D. degrees in Computer Science from Columbia University, and a B.S. in Computer Science and in Mathematics and Statistics from the University of Massachusetts at Amherst.



Box Covers and Domain Orderings for Beyond Worst-Case Join Processing

Kaleb Alway
University of Waterloo, Canada

Eric Blais
University of Waterloo, Canada

Semih Salihoglu
University of Waterloo, Canada

Abstract

Recent *beyond worst-case optimal* join algorithms Minesweeper and its generalization Tetris have brought the theory of indexing and join processing together by developing a geometric framework for joins. These algorithms take as input an index $\mathcal{B}$, referred to as a *box cover*, that stores *output gaps* that can be inferred from traditional indexes, such as B+ trees or tries, on the input relations. The performances of these algorithms highly depend on the *certificate* of $\mathcal{B}$, which is the smallest subset of gaps in $\mathcal{B}$ whose union covers all of the gaps in the output space of a query Q. Different box covers can have different size certificates and the sizes of both the box covers and certificates highly depend on the ordering of the domain values of the attributes in Q. We study how to generate box covers that contain small size certificates to guarantee efficient runtimes for these algorithms. First, given a query Q over a set of relations of size N and a fixed set of domain orderings for the attributes, we give a $\widetilde{O}(N)$-time algorithm called *GAMB* which generates a box cover for Q that is guaranteed to contain the smallest size certificate across any box cover for Q. Second, we show that finding a domain ordering to minimize the box cover size and certificate is NP-hard through a reduction from the *2 consecutive block minimization problem* on boolean matrices. Our third contribution is a $\widetilde{O}(N)$-time approximation algorithm called *ADORA* to compute domain orderings, under which one can compute a box cover of size $\widetilde{O}(K^r)$, where K is the minimum box cover for Q under any domain ordering and r is the maximum arity of any relation. This guarantees certificates of size $\widetilde{O}(K^r)$. We combine ADORA and GAMB with Tetris to form a new algorithm we call *TetrisReordered*, which provides several new beyond worst-case bounds. On infinite families of queries, TetrisReordered's runtimes are unboundedly better than the bounds stated in prior work.

2012 ACM Subject Classification Information systems → Database query processing; Theory of computation → Database query processing and optimization (theory)

Keywords and phrases Beyond worst-case join algorithms, Tetris, Box covers, Domain orderings

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.3

Related Version *Full Version*: `https://arxiv.org/abs/1909.12102` [4]

1 Introduction

Performing the natural join of a set of relational tables is a core operation in relational database management systems. After the celebrated result of Atserias, Grohe and Marx [5] that provided a tight bound on the maximum (or worst-case) size of natural join queries, now known as the *AGM bound*, a new class of *worst-case optimal* join algorithms were introduced whose runtimes are asymptotically bounded by the AGM bound. More recently, Ngo et al. and Abo Khamis et al., respectively, introduced the Minesweeper [24] algorithm, and its generalization Tetris [1, 2], which adopt a geometric framework for joins and provide *beyond worst-case* guarantees that are closer to the highest algorithmic goal of instance optimality. Henceforth, we focus on the Tetris algorithm, the more general of these two algorithms.

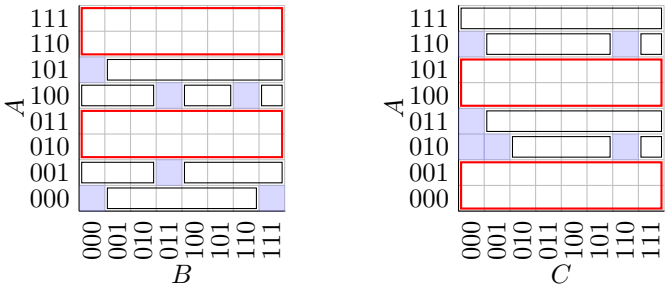


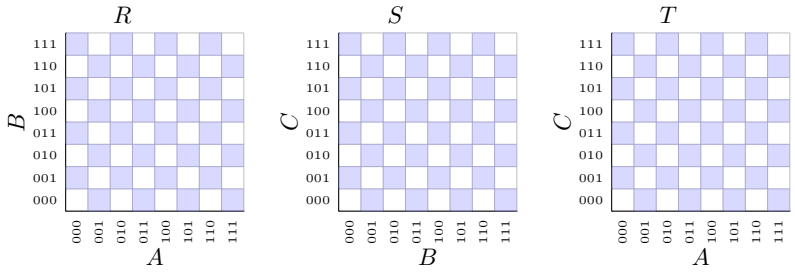
Figure 1 Box cover and certificates of the query $R(A, B) \bowtie S(A, C)$. Red boxes form a box certificate. Red and black boxes together form a box cover.

Let Q be a query over m relations $\mathcal{R}$ and n attributes $\mathcal{A}$. Let N be the total number of tuples in $\mathcal{R}$. Throughout this paper, to match the notation of reference [1], we use $\widetilde{O}$-notation to hide polylogarithmic factors in N as well as the query dependent factors m and n. Unlike traditional join algorithms that operate on input tuples, Tetris takes as input a *box cover* $\mathcal{B} = \cup_{R \in \mathcal{R}} \mathcal{B}_R$, where each $\mathcal{B}_R$ is a set of *gap boxes* (i.e., tuple-free regions) of the relation R whose union *covers* the complement of R. These boxes imply regions in the output space of queries where output tuples cannot exist. Tetris operates on these gaps by performing *geometric resolutions*, which generate new gap boxes. The runtime of Tetris is bounded by $\widetilde{O}\big(\big(C_\square(\mathcal{B})\big)^{w+1} + Z\big)$[1] where: (i) $C_\square(\mathcal{B})$ is the size of the *box certificate* for $\mathcal{B}$, which is the smallest subset of boxes in $\mathcal{B}$ that cover the gaps in the output, i.e., the complement of the output tuples of the join; (ii) w is the *treewidth* of Q; and (iii) Z is the number of output tuples. Figure 1 shows an example of this geometric framework on query $R(A, B) \bowtie S(A, C)$. Purple unit boxes indicate input tuples, the boxes in the box cover are shown with rectangles, and the boxes in the certificate are drawn as red rectangles. This Tetris result is analogous to Yannakakis's data-optimal algorithm for acyclic queries and its combination with worst-case optimal join algorithms, which yields results of the form $\widetilde{O}(N^{\text{fhtw}} + Z)$, where fhtw is the *fractional hypertree width* [15] and N is the number of tuples in the input. The performance of Tetris's results can be significantly better than Yannakakis-based algorithms, as the certificates are always $\widetilde{O}(N)$ and can be $o(N)$, e.g. constant size, on some inputs.

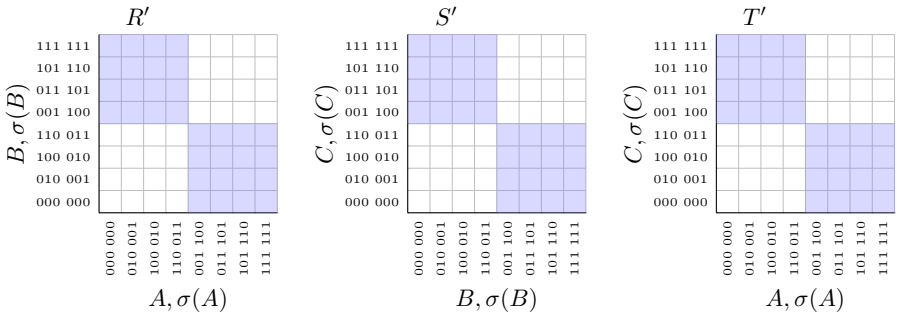
In references [1] and [24], a box cover was assumed to be inferred from the available indexes on the relations. Consider a B+ tree index on a relation $R(A, B)$ with sort order (A, B) and two consecutive tuples (a_1, b_1) and (a_1, b_2).[2] From these two tuples, a system can infer a gap box $(a_1, [b_1 + 1, b_2 - 1])$ in the output space of any join query that involves R. The boxes in Figure 1 are inferred from B+ tree indexes on R and S with sort orders (A, B) and (A, C), respectively. Using different indexes can result in box covers with vastly different certificate sizes. This motivates the first question we study in this paper:
Question 1: How can a system efficiently generate a good box cover for a set of relations?

Given a query Q, let $C_\square(Q)$ be the minimum certificate size across all possible box covers for the relations in Q.[3] An ideal goal for a system would be to efficiently generate a box cover whose certificate is of size $C_\square(Q)$, ensuring performance as a function of $C_\square(Q)$. We refer to this problem as `BoxMinC`. We present a surprisingly positive result for `BoxMinC`:

[1] A second upper bound that depends on the number of attributes instead of w is also provided in [1].
[2] This example is borrowed from reference [1].
[3] Note that our use of the notation $C_\square(Q)$ is different from reference [1], where $\mathcal{B}$ was assumed to be given, and $C_\square(Q)$ was used to indicate the certificate size for $\mathcal{B}$. Since we drop this assumption, $C_\square(\mathcal{B})$ here denotes the certificate size for $\mathcal{B}$ and $C_\square(Q)$ denotes the certificate size over all possible box covers.



(a) $Q = R \bowtie S \bowtie T$.



(b) $Q' = R' \bowtie S' \bowtie T' = \sigma(R) \bowtie \sigma(S) \bowtie \sigma(T)$.

Figure 2 Two equivalent queries (up to attribute reorderings) with different box certificate sizes.

▶ **Theorem 1.** *Given a database D, there is a $\widetilde{O}(N)$-time algorithm that can generate a box cover $\mathcal{B}$ of size at most $\widetilde{O}(N)$ that contains a certificate of size $\widetilde{O}(C_\square(Q))$ for any join query Q over any subset of relations in D.*

Therefore, in $\widetilde{O}(N)$ time and space, a system can generate a globally good box cover (an index) for all possible join queries over a database.[4] We achieve this result by observing that the set of all maximal gap boxes in the complements of the relations contains a certificate of size $|C_\square(Q)|$ and we provide an $\widetilde{O}(N)$-time algorithm called *GAMB* that **g**enerates **a**ll **m**aximal dyadic gap **b**oxes (and possibly some non-maximal ones) from the relations.

In the second question we study, we consider evaluating a single query Q. There are simple queries which can be geometrically complex and require large box covers and certificates. In many cases, these queries can be modified by reordering each attribute's domain so that smaller covers and certificates are possible. Figure 2 shows an example. In the example, the queries Q and Q' are both triangle queries joining three binary relations. These queries are equivalent up to reordering the domains of each attribute. That is, it is possible to reorder the rows and columns of the grid in Figure 2a to obtain Figure 2b. Let σ be the set of three permutations on the domains of A, B, and C which transforms Q into Q'. Specifically, for each attribute, σ maps the even values to values between 000 and 011, and the odd values to values between 100 and 111. Despite their equivalence up to reorderings, Q requires a box cover of size 96, as each white grid cell in Figure 2a must have a unit gap box covering

[4] Since any system has to spend $\Omega(N)$ time to index its tuples, this time is within an $\widetilde{O}(1)$ factor of any other indexing approach. Appendix B shows that a box cover index can also be maintained efficiently. The Tetris runtimes in reference [1] do not add a $\widetilde{O}(N)$ indexing component because it is assumed that indexes are given. In practice, this cost must be paid at some point by the database to answer queries.

it, while Q' only requires a box cover size 6. The same also applies to the certificate sizes, as every gap box in the box cover must also be part of the box certificate in this case. By extending the domains of the attributes, the difference in box cover and certificate sizes can be made arbitrarily large. Therefore, a system could improve the performance of Tetris significantly by reordering the domains of attributes. This motivates our second question:
Question 2: How can a system efficiently reorder the domains to obtain a small box cover?

We refer to the problem of finding a domain ordering σ such that the minimum box cover size under σ is minimized as $\texttt{DomOr}_{\texttt{BoxMinB}}$. Let $\mathcal{B}^*$ be the minimum size box cover for a query under any domain ordering, $K = |\mathcal{B}^*|$, and σ^* be the ordering under which $\mathcal{B}^*$ is achieved. We first provide a hardness result showing that computing σ^* is NP-hard through a reduction from the *2 consecutive block minimization problem* on boolean matrices [17]. We then provide an approximation algorithm, which we refer to as *ADORA*, for **A**pproximate **D**omain **Or**dering **A**lgorithm, to obtain the following result:

▶ **Theorem 2.** *Let r be the maximum arity of any relation in the query Q and let K be the minimum box cover size for Q under any domain ordering. There is a $\widetilde{O}(N)$-time algorithm that computes a domain ordering σ for Q, under which one can compute a box cover of size $\widetilde{O}(K^r)$, guaranteeing a certificate of size $\widetilde{O}(K^r)$.*

After σ is obtained with ADORA, a system can run GAMB, which has the same asymptotic runtime, to obtain a box cover that guarantees certificates of size $\widetilde{O}(K^r)$. ADORA is based on an intuitive and powerful heuristic that groups the domain values in an attribute that have identical value combinations in the remaining attributes across the relations and makes the values in each group consecutive. Our approximation ratio does not depend on any other parameters of the query, such as different notions of width or the number of relations. Once an ordering is obtained, Tetris can be executed on the reordered query and the results converted back to the original domain. This technique is formalized in our algorithm *TetrisReordered*. We construct families of queries for which Tetris on a default ordering has a polynomial runtime with an arbitrarily high degree, but for which TetrisReordered runs in $\widetilde{O}(N)$ time.

2 Notation and Preliminaries

Throughout this paper, we work with a fixed database D. A *query* Q is an equi-join over a set of m fixed relations $\mathcal{R}$ and a set of n attributes $\mathcal{A}$ from D. We do not differentiate between a query and a query instance, so Q refers to the instance of Q in D. As in reference [1], for ease of presentation we assume the domains of each attribute $A \in \mathcal{A}$ consist of all d bit integers but our results only require domain values to be discrete and ordered. For $R \in \mathcal{R}$ and $A \in \mathcal{A}$, the attribute set of R is denoted $\text{attr}(R)$ and the domain of A is denoted $\text{dom}(A)$.

Tetris takes as input a box cover $\mathcal{B}$ that contains *dyadic* gap boxes, which are boxes whose span over each attribute is encoded as a binary prefix. Let $R \in \mathcal{R}$ contain n_R attributes. Formally, a dyadic gap box in $\mathcal{B}_R$ is an n_R-tuple $b = \langle s_1, s_2, \ldots, s_{n_R} \rangle$ where each s_i is a binary string of length at most d. We use $*$ to denote the empty string. We sometimes use $b.A$ to denote the prefix in b corresponding to attribute A. For example, if d is 3, the dyadic box $\langle 01, 1 \rangle$ for $R(A_1, A_2)$ is the box whose A_1 and A_2 dimensions include all values with prefix 01 and 1, respectively, i.e., it is the rectangle with sides $\langle [010 - 011], [100 - 111] \rangle$. Using dyadic boxes allows Tetris to perform *geometric resolutions* (explained momentarily) efficiently, which is needed to prove the runtime bounds of Tetris.

Although the details of how Tetris works are not necessary to understand our techniques and contributions, we give a brief overview as background and refer the reader to reference [1] for details. Assume each box in $\mathcal{B}$, say those coming from $\mathcal{B}_R$, are extended, with prefix $*$,

to every attribute not in attr(R). This allows us to think of $\mathcal{B}$ as a single gap box index over the output space. The core of Tetris is a recursive subroutine that determines whether the set of boxes in $\mathcal{B}$ covers the entire n-dimensional output space $\langle *, *, \ldots, * \rangle$ and returns either YES or NO with an output tuple o as a witness. The witnesses are inserted into $\mathcal{B}$. During the execution, this subroutine performs *geometric resolutions* that take two boxes that are adjacent in one dimension and construct a new box that consists of the union of the intervals in this dimension (and the intersection in all others). When boxes are dyadic, geometric resolution can be done in $\widetilde{O}(1)$ time. This recursive subroutine is called as many times as there are output tuples until it finally returns YES. Two variants of Tetris, called Tetris-Preloaded and Tetris-LoadBalanced run in time $\widetilde{O}(C_\square(\mathcal{B})^{w+1} + Z)$ and $\widetilde{O}(C_\square(\mathcal{B})^{n/2} + Z)$, respectively (see Theorems 4.9 and 4.11 in reference [1]). $C_\square(\mathcal{B})$ in Tetris's runtime is the box certificate size of $\mathcal{B}$, which is the size of the smallest subset $\mathcal{B}'$ of $\mathcal{B}$, such that the union of boxes in $\mathcal{B}'$ and the union of boxes in $\mathcal{B}$ cover exactly the same space. Equivalently, $C_\square(\mathcal{B})$ is the size of the smallest subset $\mathcal{B}'$ of $\mathcal{B}$ whose extended boxes (with $*$'s as described above) cover all of the gaps in the output space.

We end this section with a note on dyadic vs. general boxes. The notions of certificate, box cover, and the problems we study can be defined in terms of dyadic or general boxes. Except in Section 4, the term box refers to general boxes, and our optimization problems are defined over general box covers and certificates. For both certificates and box covers, the minimum size obtained with dyadic boxes and general boxes are within $\widetilde{O}(1)$ of each other. This is because a dyadic box is a general box by definition and any general box can be partitioned into $\widetilde{O}(1)$ dyadic boxes (Proposition B.14 in reference [1]). Our approximation results for general boxes imply approximation results for dyadic boxes up to $\widetilde{O}(1)$ factors. However, a hardness result for one version does not imply hardness of the other. Our hardness results apply only to general boxes. However, we use dyadic boxes extensively because they are a powerful analytical tool which the results of this paper and reference [1] rely on.

3 Related Work

3.1 Box Cover Problems

The complement of a relation R with k attributes can be represented geometrically as a set of axis-aligned, rectilinear polytopes in k-dimensional space, which may have holes (the tuples in R form the exteriors of the polytopes). The number of vertices in these polytopes is bounded (up to a constant factor) by the number of tuples in the relation.Therefore our work is closely related to covering rectilinear polytopes with a minimum number of rectangles in geometry. This problem has been previously studied in the 2-dimensional setting, i.e., for polygons. The problem is known to be NP-complete, even when the polygon is hole-free [10] and MaxSNP-hard for polygons with holes [6]. There are several approximation algorithms for the problem. Franzblau [12] designed an algorithm that approximates the optimal solution to a factor of $O(\log n)$, where n is the number of vertices in the polygon. If the polygon is hole-free, the approximation factor improves to 2. Anil Kumar and Ramesh [20] showed a tighter approximation ratio of $O(\sqrt{\log n})$ for the same algorithm on polygons with holes. Franzblau et al. [13] also showed the problem is solvable in polynomial time in the special case when polygons are vertically convex. All of these results are limited to 2D and little is known about the problem in higher dimensions.

The approximation algorithms above can be used to generate box covers for the complement of a binary relation R. This is a special case of `BoxMinC`, where the input is a trivial query with a single binary relation R. Outside of this limited setting, the connection

of covering axis-aligned and rectilinear polygons to `BoxMinC` breaks. This is because the certificate of a query in this case is the smallest number of boxes that cover the complement of the output, using boxes from the relations. In this case, because the output is not yet computed, it is not known a priori which polytopes should be covered.

There are variants of covering polygons that are less directly related to our problems. Reference [16] studies the more general problem of covering polygons with only obtuse interior angles, and provides approximation algorithms. Reference [21] studies covering the input polygon with squares instead of rectangles. For a survey of geometric covering and packing problems, including shapes beyond polytopes, we refer the reader to references [9] and [28].

3.2 Orderings in Matrices

There are several problems related to ordering the rows and columns of boolean matrices to achieve different optimization goals. The closest to our work is the consecutive block minimization problem (CBMP) [19]. Our hardness results are based on a variant of CBMP, called 2 consecutive block minimization [17], which we review in Section 5.1. There are two other ordering problems for boolean matrices, which are less related to our work: (i) the *consecutive ones property test* determines whether there is a column ordering such that each row has only one consecutive block of ones [8]; (ii) the *doubly lexical ordering problem* finds a row and column ordering such that both rows and columns are in lexicographic order [22]. Both problems have polynomial time solutions.

3.3 Worst-Case and Beyond Worst-Case Join Algorithms

A join algorithm is said to be *worst-case optimal* if it runs in time $\widetilde{O}(\text{AGM}(Q))$, where the AGM bound [5] is the worst-case upper bound on the number of output tuples for a query based on its shape and the number of input tuples. Examples of worst-case optimal join algorithms are Leapfrog Triejoin [29], NPRR [25], and Generic Join [26]. A survey on worst-case optimal join algorithms can be found in reference [23]. There are several results that consider other properties of the query and provide worst-case upper bounds on the size of query outputs that are better than the AGM bound. Olteanu and Závodný [27] show that worst-case sizes of queries in *factorized representations* can be asymptotically smaller than the AGM bound and provide algorithms that meet these factorized bounds. Joglekar and Ré [18] developed an algorithm which provides degree-based worst-case results that assume knowledge of degree information for the values in the query. Similarly, references [3] and [14] provide worst-case bounds based on information theoretical bounds that take into account, respectively, more general degree constraints and functional dependencies.

Several results go beyond worst-case bounds and are closer to the notion of instance optimality. The earliest example is Yannakakis' data-optimal algorithm [30] for acyclic queries that runs in time $O(N + Z)$. This was later generalized to an algorithm [11] for arbitrary queries which runs in time $\widetilde{O}(N^{fhtw} + Z)$, where $fhtw$ is the query's *fractional hypertree width* [15]. The Minesweeper algorithm [24] developed the measure of *comparison certificate* C_{comp} for comparison-based join algorithms, which captures the minimum number of comparisons needed to prove the output of a join query is correct. Minesweeper runs in time $\widetilde{O}(|C_{comp}|^{w+1} + Z)$, where Z is the number of output tuples and w is the query's treewidth. The Tetris algorithm [1], which motivates our work, generalizes comparison certificates to the geometric notion of a box certificate, reviewed in Section 1. For every comparison certificate C_{comp}, there is a box certificate of size at most $|C_{comp}|$. In this sense, box certificates are

Algorithm 1 GAMB(R): Generates all maximal dyadic gap boxes of R.

```
1: B := ∅, B̄ := ∅
2: for t ∈ R do
3:     for every dyadic box b such that t ∈ b do
4:         B̄ := B̄ ∪ {b}
5:         for A ∈ attr(R) such that b.A ≠ ∗ do
6:             Let b′ be the box when the last bit of b.A is flipped
7:             B := B ∪ {b′}
8: return B \ B̄
```

stronger than comparison certificates, and Tetris subsumes the certificate-based results of Minesweeper. Our results on finding box covers with small certificates and domain orderings with small box covers improve the bounds provided by Tetris.

4 Generating a Box Cover

Since the runtime of Tetris depends on the certificate size of its input box cover, an important preprocessing step for the algorithm is to generate a box cover with a small certificate. Ideally, a system should generate a box cover that contains a certificate of minimum size, across all box covers. We defined this quantity as $C_\square(Q)$ in Section 1. The following lemma states two facts about dyadic boxes that are crucial for our results and the results in reference [1].

▶ **Lemma 3.** *(Propositions B.12 and B.14 [1]) Let b be any dyadic box. Then there are $\widetilde{O}(1)$ dyadic boxes which contain b. Let b' be any (not necessarily dyadic) box. Then b' can be partitioned into a set of $\widetilde{O}(1)$ disjoint dyadic boxes whose union is equal to b'.*

Let a dyadic gap box b for a relation R be *maximal* if b cannot be enlarged in any of its dimensions and still remain a dyadic gap box, i.e., not include an input tuple of R. Generating a box cover with certificate size $\widetilde{O}(C_\square(Q))$ can be done by generating the set of all maximal dyadic gap boxes in the input relations. This is because: (1) any general box can be decomposed into $\widetilde{O}(1)$ dyadic boxes by Lemma 3, so decomposing a general box cover into a dyadic one can increase its certificate size by at most a factor of $\widetilde{O}(1)$; and (2) expanding any non-maximal dyadic boxes to make them maximal can only decrease the size of the certificate. We will show that given any query Q with N input tuples, we can generate all maximal dyadic gap boxes over all of the relations in Q in $\widetilde{O}(N)$ time. This also implies that the number of maximal dyadic boxes is $\widetilde{O}(N)$. Interestingly, this is not true for general gap boxes, of which there can be a super-linear number (see Appendix A for an example).

Algorithm 1 shows the pseudocode for our algorithm *GAMB* that **g**enerates **a**ll **m**aximal dyadic gap **b**oxes for a relation R in $\widetilde{O}(N)$ time. GAMB loops over each dyadic box b covering each tuple t in R, explores boxes that are adjacent to b (which may or may not be gap boxes) and inserts these into a set B. Then it subtracts the set of all dyadic boxes covering any tuples from B to obtain a set of gap boxes. As we argue, this set contains every maximal dyadic gap box (and possibly some non-maximal ones). To generate all maximal boxes for a query $Q = (\mathcal{R}, \mathcal{A})$, we can simply iterate over each $R \in \mathcal{R}$ and invoke GAMB.

▶ **Theorem 4.** *GAMB generates all maximal dyadic gap boxes of a relation R in $\widetilde{O}(N)$ time.*

Proof. Let b' be a maximal dyadic gap box for R. Let A be an attribute of R for which b' specifies at least one bit (so $b'.A \neq *$). Let b be the dyadic box obtained from b' by flipping the last bit of $b'.A$. Since b' is maximal, b contains at least one tuple $t \in R$. Since b is a

dyadic box containing t, some iteration of the for-loop on line 3 will reach box b. Then the for-loop on line 5 at some iteration will loop over A and generate exactly b' on line 6. Thus b' is added to B and since b' is a gap box, GAMB will not add it to $\overline{B}$ (which only contains non-gap boxes). Therefore b' will be in the output of GAMB. Note that the returned set does not contain any non-gap boxes of R, since every box which contains any tuple of R is added to $\overline{B}$. The outer-most for loop has N iterations. The for loop on line 3 has $\widetilde{O}(1)$ iterations by Lemma 3. The for-loop on line 5 has n, so $\widetilde{O}(1)$, iterations. Finally, the set difference on line 8 can be done by sorting both B and $\overline{B}$ and iterating lockstep through the sorted boxes. Therefore, the total runtime of GAMB is $\widetilde{O}(N)$. ◀

By our earlier observation based on Lemma 3, running GAMB as a preprocessing step is sufficient to generate a box cover with a certificate of size $\widetilde{O}(C_\square(Q))$. Combined with runtime upper bounds of Tetris from reference [1], we can state the following corollary:

▶ **Corollary 5.** *Given a database D of relations with N total tuples, in $\widetilde{O}(N)$ preprocessing time, one can generate a box cover $\mathcal{B}$ such that running Tetris on $\mathcal{B}$ yields $\widetilde{O}\big(\big(C_\square(Q)\big)^{w+1}+Z\big)$ or $\widetilde{O}\big(\big(C_\square(Q)\big)^{n/2}+Z\big)$ runtimes for any query Q over D.*

One interpretation of this result is that in $\widetilde{O}(N)$ preprocessing time, a system can generate *a single global index* that will make Tetris efficient on all possible join queries over a database D. In fact, using the bounds from reference [1], these are the best bounds we can obtain up to an $\widetilde{O}(1)$ factor when Q is fixed, since $C_\square(Q)$ is the minimum certificate size for any box cover of Q. This is surprisingly achieved with the same index for all queries, so the $\widetilde{O}(N)$ preprocessing cost need only be incurred once for a workload of any number of joins. To improve on these bounds, we must modify Q to reduce the box certificate size. We next explore domain orderings as a method to improve these bounds.

5 Domain Ordering Problems

We next study the $\texttt{DomOr}_{\texttt{BoxMinB}}$ problem. Given a query Q, our goal is to find the minimum size box cover possible under any *domain ordering* for Q and to find the domain ordering σ^* that yields this minimum possible box cover size. We begin by defining a domain ordering.

▶ **Definition 6** (Domain ordering)**.** *A* domain ordering *for a query $Q = (\mathcal{R}, \mathcal{A})$ is a tuple of $|\mathcal{A}|$ permutations $\sigma = (\sigma_A)_{A\in\mathcal{A}}$ where each σ_A is a permutation of dom(A).*

▶ **Example 7.** Let A and B be attributes over 2-bit domains. Let $R(A, B)$ be the following relation presented under the default domain ordering $[00, 01, 10, 11]$ for both A and B:

$$R(A, B) = \big\{\langle 00, 00\rangle, \langle 01, 11\rangle, \langle 10, 00\rangle, \langle 11, 11\rangle\big\}$$

Consider the domain ordering σ where $\sigma_A{=}\sigma_B{=}\{00 \mapsto 00,\ 01 \mapsto 10,\ 10 \mapsto 11,\ 11 \mapsto 01\}$. We write σ as $\sigma_A = \sigma_B = [00, 11, 01, 10]$ to indicate the new "locations" of the previous domain values in the new ordering. Then $\sigma(R)$ denotes the following relation:

$$\sigma(R)(A, B) = \big\{\langle 00, 00\rangle, \langle 10, 01\rangle, \langle 11, 00\rangle, \langle 01, 01\rangle\big\}$$

The choice of domain ordering can have a significant effect on box cover sizes and their certificates. We show in Section 5.3 that given a query Q over n attributes, we can construct an infinite family of queries over n attributes which require arbitrarily large box covers and certificates under a default domain ordering, but under another domain ordering, have box covers and certificates of the same size as Q. Our specific problem is this:

▶ **Definition 8** ($\texttt{DomOr}_{\texttt{BoxMinB}}$). *Let $K_\square(\sigma(Q))$ be the minimum box cover size one can obtain for the query $\sigma(Q)$ obtained from Q by ordering the domains according to σ. Given a query Q, output a domain ordering σ^* such that $K_\square(\sigma^*(Q)) = \min_\sigma K_\square(\sigma(Q))$.*

In Section 5.1, we show that $\texttt{DomOr}_{\texttt{BoxMinB}}$ is NP-hard. In Section 5.2, we present ADORA, an approximation algorithm for $\texttt{DomOr}_{\texttt{BoxMinB}}$. Section 5.3 combines ADORA, GAMB, and Tetris in an algorithm we call *TetrisReordered*, which has new beyond worst-case bounds. In Section 5.3 we also present infinite classes of queries for which TetrisReordered runs unboundedly faster than the versions of Tetris from reference [1].

5.1 $\texttt{DomOr}_{\texttt{BoxMinB}}$ is NP-hard

Our reduction is from the *2 consecutive block minimization problem (2CBMP)* on boolean matrices [17].[5] In a boolean matrix M, a *consecutive block* is a maximal consecutive run of 1-cells in a single row of M, which is bounded on the left by either the beginning of the row or a 0-cell, and bounded on the right by either the end of the row or a 0-cell. We use $\text{cb}(M)$ to denote the total number of consecutive blocks in M over all rows. Let M be a boolean matrix stored as a 2D dense array, each row of which contains at most 2 1-cells. 2CBMP is the problem of finding an ordering σ_c^* on the columns of M such that $\text{cb}(\sigma_c^*(M)) = \min_{\sigma_c} \text{cb}(\sigma_c(M))$. 2CBMP was shown to be NP-hard in reference [17].

▶ **Theorem 9.** *$\texttt{DomOr}_{\texttt{BoxMinB}}$ is NP-hard.*

Proof. We focus on the special case where Q contains a single relation $R(A, B)$ over exactly 2 attributes and show that $\texttt{DomOr}_{\texttt{BoxMinB}}$ is NP-hard even in this case. This implies $\texttt{DomOr}_{\texttt{BoxMinB}}$ is NP-hard for any number of attributes and relations, since one can duplicate R to another relation S with the same schema, and extend R and S to a third attribute C, taking $R' = R \times \text{dom}(C)$. The ordering that solves $\texttt{DomOr}_{\texttt{BoxMinB}}$ on $R' \bowtie S'$ (a trivial intersection query) also minimizes the box cover size for R. For the purposes of the proof, we model R as a boolean matrix M', with a row for each value in $\text{dom}(B)$ and a column for each value in $\text{dom}(A)$. Each cell of the matrix corresponds to a possible tuple in $\text{dom}(A) \times \text{dom}(B)$. The matrix M' contains a 0-cell in column i and row j if the tuple $t = \langle i, j \rangle \in R$, and a 1-cell otherwise. This means that a box cover $\mathcal{B}$ for R corresponds directly to a set of rectangles which cover all of the 1-cells of M', and vice-versa. Readers can assume M' is given to $\texttt{DomOr}_{\texttt{BoxMinB}}$ as a dense matrix or a list of tuples, i.e., (i, j) indices for the 0 cells.

Let M be an $n \times m$ boolean matrix input to 2CBMP. We construct a $(4n) \times (m+2n)$ matrix M' for input to $\texttt{DomOr}_{\texttt{BoxMinB}}$. For each row r_i of M, we create 4 rows in M': $r_{i,1}, r_{i,2}, p_{i,1}$, and $p_{i,2}$. $r_{i,1}$ and $r_{i,2}$ are duplicates of the original row r_i, and $p_{i,1}$ and $p_{i,2}$ are the *padding rows* of r_i. We also add 2 *padding columns* that contain 1-cells in the 4 rows of r_i and $2n-2$ columns that contain only 0-cells for the 4 rows for of r_i. Let S_i be the set of columns with 1-cells in row r_i of M. Let e_S be the row vector of length $m + 2n$ with value 1 on all indices in $S \subseteq [m + 2n]$, and 0 everywhere else. The new rows are defined as: (i) $p_{i,1} = e_{\{m+2i-1\}}$; (ii) $r_{i,1} = e_{S_i \cup \{m+2i-1\}}$; (iii) $r_{i,2} = e_{S_i \cup \{m+2i\}}$; and (iv) $p_{i,2} = e_{\{m+2i\}}$. We insert these rows in the (i)-(iv) order, for $r_1, ..., r_n$, and refer to this order as the *default row ordering* of M'. We refer to the column ordering of M' after this transformation as the *default column ordering* of M'. An example transformation from M to M' is shown in Figure 3.

[5] In this section, we use the informal convention of discussing the NP-hardness of minimization problems. Since NP-hardness is defined for decision problems, when we state that a minimization problem like 2CBMP is NP-hard, we are implicitly referring to the decision problem which takes as input an additional positive integer k and accepts if and only if the minimum value of the objective function is at most k.

$$M = \begin{matrix} r_1 \\ r_2 \end{matrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad M' = \begin{matrix} p_{1,1} \\ r_{1,1} \\ r_{1,2} \\ p_{1,2} \\ p_{2,1} \\ r_{2,1} \\ r_{2,2} \\ p_{2,2} \end{matrix} \left[\begin{array}{cccc|cc|cc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Figure 3 An example of the 2CBMP input matrix M and its corresponding M' matrix.

To prove this theorem, it suffices to prove that there exists an ordering σ_c on the columns of M such that $\text{cb}(\sigma_c(M)) \leq k$ if and only if there exist orderings $\sigma' = (\sigma'_r, \sigma'_c)$ on the rows and columns of M' such that $\sigma'(M')$ admits a box cover of size $\leq k + 2n$. Proving one direction of this claim is simple. If there exists an ordering σ_c on the columns of M such that $\text{cb}(\sigma_c(M)) \leq k$, then set σ'_r equal to the default row ordering of M'. Also, set the last $2n$ columns in σ'_c equal to the default column ordering of the last $2n$ columns of M'. Then, set the first m columns in σ'_c equal to σ_c. Then, the 1-cells in the first m columns of $\sigma'(M')$ can be covered by at most k boxes, and the 1-cells in the last $2n$ columns can be covered by $2n$ boxes, for a total box cover size of at most $k + 2n$.

Proving the converse is significantly more involved. Let $\sigma' = (\sigma'_r, \sigma'_c)$ be an ordering on the rows and columns of M' such that $\sigma'(M')$ admits a box cover B of size $\leq k+2n$. We start with two definitions. Two rows $r_{i,j}$ and $r_{k,\ell}$ in M' ($i, k \in [n]$ and $j, \ell \in \{1, 2\}$) are *equivalent* if r_i and r_k are equal rows in M (ie. r_i and r_k have 1-cells in the same columns in M). A *run* of equivalent rows is a sequence E of one or more $r_{i,j}$ rows which are consecutive in σ'_r such that all rows in E are equivalent to one another. We show a sequence of 6 steps that transform σ' to match the default row ordering and except in the first m columns also the default column ordering. For each step, we prove that we can reorder σ' without increasing the number of boxes such that a claim is true of the reordered $\sigma'(M')$, assuming that all of the previous claims hold. The proofs of these claims are provided in Appendices C.1-C.6.

1. Every $r_{i,j}$ row can be made adjacent to some equivalent $r_{k,\ell}$ row. (App. C.1)
2. Every run of equivalent $r_{i,j}$ rows can be made to have even length. (App. C.2)
3. Every run of equivalent $r_{i,j}$ rows can be made to have length 2. (App. C.3)
4. The padding rows $p_{i,j}$ can be made adjacent to their matching $r_{i,j}$ rows. (App. C.4)
5. The row order σ'_r can be made to exactly match the default row order of M'. (App. C.5)
6. The column order σ'_c can be made to exactly match the default column order of M' on the last $2n$ columns. (App. C.6)

Two 1-cells c_1, c_2 are *independent* in $\sigma'(M')$ if there is no box containing c_1, c_2 that contains only 1-cells. An *independent set* is a set of pairwise independent 1-cells. An independent set in $\sigma'(M')$ of size S implies that the minimum box cover size of $\sigma'(M')$ is at least S. We will proceed by constructing a sufficiently large independent set in $\sigma'(M')$. After the above 6 steps, M' and $\sigma'(M')$ differ only by the ordering of the first m columns. In $\sigma'(M')$, the last $2n$ columns contain an independent set of size $2n$, by taking the single 1-cell from each of the $p_{i,j}$ rows. These $2n$ 1-cells are independent from all 1-cells in the first m columns of σ'_c. Let σ_c be the ordering of the first m columns in σ'_c. We claim the first m columns contain an independent set of size $\text{cb}(\sigma_c(M))$. First, any two 1-cells in separate 4-row units are independent from one another, because the padding rows between them contain only 0-cells on the first m columns. If a row of $\sigma_c(M)$ has only one consecutive block, add a 1-cell from the corresponding 4-row unit to the independent set. If a row of $\sigma_c(M)$ has two consecutive blocks, there are two 1-cells in the first m columns of the corresponding 4-row unit which are

independent from one another. Add both to the independent set. Combining the independent sets from the first m columns and the last $2n$ columns, we get an independent set of size $\text{cb}(\sigma_c(M)) + 2n$. Therefore any box cover of $\sigma'(M')$ has at least $\text{cb}(\sigma_c(M)) + 2n$ boxes. We assumed $\sigma'(M')$ has a box cover of size $\leq k + 2n$, which implies $\text{cb}(\sigma_c(M)) + 2n \leq k + 2n$, so $\text{cb}(\sigma_c(M)) \leq k$, completing the reduction. ◀

5.2 Approximating $\texttt{DomOr}_{\texttt{BoxMinB}}$

In this section, we provide a $\widetilde{O}(N)$-time approximation algorithm for $\texttt{DomOr}_{\texttt{BoxMinB}}$. Section 5.2.1 develops some machinery necessary to prove our approximation ratio, and Section 5.2.2 presents our approximation algorithm, ADORA. Section 5.3 combines ADORA, GAMB, and Tetris to state new beyond worst-case bounds for join processing.

5.2.1 Dividing Relations into Hyperplanes

In the simplest case, suppose the best domain ordering σ^* for Q yields a minimum box cover of size 1. Then there is a single gap box b in some relation $\sigma^*(R)$ such that $\mathcal{B} = \{b\}$ forms a box cover for $\sigma^*(Q)$. Fix an arbitrary attribute $A \in \text{attr}(R)$. We can partition $\text{dom}(A)$ into two sets: values which are in the A-range spanned by b, and values which are not. Consider the domain ordering σ_A obtained by placing all the domain values spanned by b first (in any order), followed by all other values. Doing this for each $A \in \mathcal{A}$ yields a domain ordering σ which recovers the box b to attain a box cover of size 1. Intuitively, any domain values for A which lie in the span of the same set of boxes in the minimum box cover should be placed next to one another. This can be generalized to an approximation algorithm for any minimum box cover size. We begin with definitions needed to formalize this approach.

▶ **Definition 10** (A-hyperplane). *Let $R \in \mathcal{R}$ be over a set of attributes $attr(R)$. Let $A \in attr(R)$ and $a \in dom(A)$. The A-hyperplane of R defined by a is the relation $H(R, A, a) = \pi_{attr(R)\setminus\{A\}}(\sigma_{A=a}(R))$ if $|attr(R)| > 1$ and $H(R, A, a) = \{|\sigma_{A=a}(R)|\}$ if $|attr(R)| = 1$.*

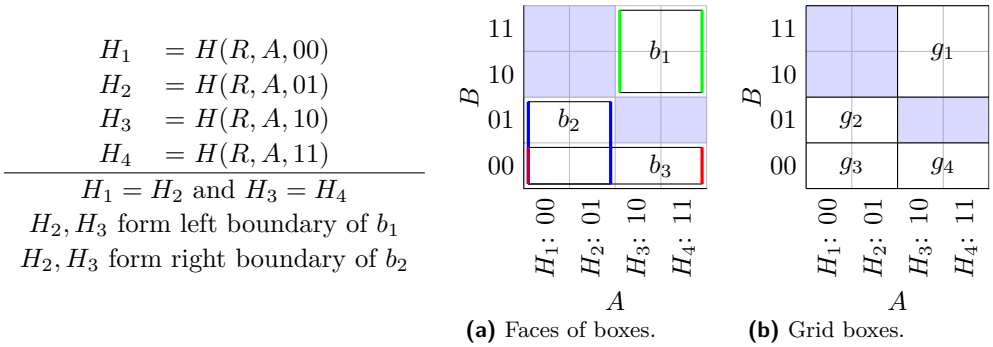
Let $n_R = |\text{attr}(R)|$. The A-hyperplane defined by a in R can be thought of as the "slice" of the n_R-dimensional space occupied by R containing only the (n_R-1)-dimensional subspace where the attribute A is fixed to the value a. This is a natural generalization of "rows" and "columns" which were useful for discussing 2-dimensional relations in Section 5.1.

▶ **Definition 11** (Equivalent domain values). *Let $Q = (\mathcal{R}, \mathcal{A})$ be a query, let $A \in \mathcal{A}$, and let $a_1, a_2 \in dom(A)$. a_1 and a_2 are* equivalent in Q *if for all $R \in \mathcal{R}$ we have $H(R, A, a_1) = H(R, A, a_2)$. In this case, we write $a_1 \sim a_2$.*

For $a \in \text{dom}(A)$, the subset of domain values $\text{Eq}(a) = \{a' \in \text{dom}(A) : a \sim a'\} \subseteq \text{dom}(A)$ is called the *equivalence class* of a. The equivalence classes for all of the values in $\text{dom}(A)$ form a partition of $\text{dom}(A)$. The next lemma bounds the number of these equivalence classes as a function of the minimum box cover size of any domain ordering σ.

▶ **Lemma 12.** *Let σ be a domain ordering for $Q = (\mathcal{R}, \mathcal{A})$. Let A be an attribute in $\mathcal{A}$ and h be the number of equivalence classes of the values in $dom(A)$. Then $h \leq 2 \cdot K_{\square}(\sigma(Q)) + 1$.*

Proof. Let $A \in \mathcal{A}$ and let $a_1, a_2 \in \text{dom}(A)$ be such that a_1 directly precedes a_2 in σ_A and $a_1 \not\sim a_2$. We refer to the a_1, a_2 boundary as a "switch" along A. Observe that there are at least $h-1$ switches along A. This minimum is attained when the values in each equivalence class are placed in a single consecutive run in σ_A. Since $a_1 \not\sim a_2$, there is some



(a) Faces of boxes. **(b)** Grid boxes.

Figure 4 An illustration of how A-hyperplane switches form the boundaries of the gap boxes of R (left) and how dividing R into grid cells defined by hyperplane switches induces a box cover (right).

relation $R \in \mathcal{R}$ such that $H_1 = H(R, A, a_1) \neq H(R, A, a_2) = H_2$. Then there is some tuple t which is in H_1 but not H_2 or vice versa. Assume w.l.o.g. that $t \in H_1$ and $t \notin H_2$. Let $t_1 = \langle a_1, t \rangle$ and $t_2 = \langle a_2, t \rangle$ be the tuples that extend t to attribute A with values a_1 and a_2, respectively. This means that $t_1 \in R$ and $t_2 \notin R$. Let $\mathcal{B}$ be a box cover for $\sigma(Q)$ with $K_\square(\sigma(Q))$ boxes. Let $\mathcal{B}_R$ be the set of boxes in $\mathcal{B}$ that are from R and cover the complement of R (so $|\mathcal{B}_R| \leq K_\square(\sigma(Q))$). Let $b \in \mathcal{B}_R$ be a box covering t_2 (and not t_1 since b is a gap box). In this context, a *face* of b along the A axis is one of the two distinct portions of the boundary of b contained in a hyperplane orthogonal to the A axis that does not contain any interior points of b. Since a_1 and a_2 are adjacent in σ_A, one face of b along the A axis is the (a_1, a_2) switch, i.e. one face of b lies on the boundary between H_1 and H_2. Every box b has exactly two faces along A, so there are $\leq 2K_\square(\sigma(Q))$ faces of boxes in $\mathcal{B}$ along the A axis. Two different switches cannot correspond to the same face of the same box. As an example, Figure 4a shows the switches in attribute A and the faces of gap boxes that these switches correspond to, which are highlighted in colour. This completes the argument that each (a_1, a_2) switch corresponds to a distinct face of some box along the A axis. There are at least $h-1$ switches and at most $2K_\square(\sigma(Q))$ box faces, so $h \leq 2K_\square(\sigma(Q))+1$. ◀

5.2.2 ADORA

Lemma 12 inspires an approximation algorithm for $\texttt{DomOr}_{\texttt{BoxMinB}}$. Let σ^* be the optimal domain ordering for $\texttt{DomOr}_{\texttt{BoxMinB}}$ on Q. Let $K = K_\square(\sigma^*(Q))$ throughout this section. Algorithm 2 presents the pseudocode for our **A**pproximate **D**omain **Or**dering **A**lgorithm (ADORA). ADORA uses Algorithm 3 as a subroutine to produce an ordering σ_A for $\text{dom}(A)$ that contains each equivalence class in $\text{dom}(A)$ as a consecutive run.

▶ **Theorem 13.** *Let $Q = (\mathcal{R}, \mathcal{A})$ be a query and σ^* be an optimal domain ordering for $\texttt{DomOr}_{\texttt{BoxMinB}}$ on Q. Let $K = K_\square(\sigma^*(Q))$. Then ADORA produces a domain ordering σ in $\widetilde{O}(N)$ time such that $K_\square(\sigma(Q)) = \widetilde{O}(K^r)$, where r is the maximum arity of a relation in $\mathcal{R}$.*

Proof. We defer the runtime analysis of ADORA to Appendix D and prove the approximation ratio here. We begin by arguing that given an attribute A, the ordering returned by Algorithm 3 places every equivalence class of $\text{dom}(A)$ in one consecutive run. The for-loop on line 5 iterates over each $a \in \text{dom}(A)$ that appears in Q and constructs an array $\mathcal{T}[a]$. $\mathcal{T}[a]$ is the result of appending the A-hyperplanes $H(R, A, a)$ for each $R \in \mathcal{S}$ in a fixed order. Line 4 sorts each relation lexicographically starting with A (notice that the order ϕ is

Algorithm 2 ADORA($Q = (\mathcal{R}, \mathcal{A})$): Computes a domain ordering.

1: **for** $A \in \mathcal{A}$ **do**
2: $\quad \sigma_A :=$ OrderAttr(Q, A)
3: **return** $\sigma = \{\sigma_A\}_{A \in \mathcal{A}}$

defined to place A first). These two facts ensure that after the for-loop on line 5 has finished, $\mathcal{T}[a_1] = \mathcal{T}[a_2]$ if and only if $a_1 \sim a_2$. The final sort of D on line 9 sorts values of $\text{dom}(A)$, say a_i and a_j, according to the lexicographic order of $T[a_i]$ and $T[a_j]$, so the lists are compared item by item from start to end, with each item compared using attribute ordering ϕ. This ensures all A values in the same equivalence class will be in one consecutive run in σ_A.

The output of ADORA is a domain ordering σ, which orders each attribute $A \in \mathcal{A}$ according to the σ_A returned by Algorithm 3. We next prove there exists a box cover for $\sigma(Q)$ of size $\widetilde{O}(K^r)$. Let $R \in \mathcal{R}$. Suppose $|\text{attr}(R)| = n_R$ and note that $n_R \leq r$. Let $A \in \text{attr}(R)$. Lemma 12 states that $\text{dom}(A)$ contains at most $2K+1$ equivalence classes, which we proved are placed consecutively in σ_A. By definition, if $a_1 \sim a_2$, then $H(R, A, a_1) = H(R, A, a_2)$. Therefore σ_A consists of a sequence of at most $2K+1$ consecutive runs of A-values where the values in each run have identical A-hyperplanes in R. This holds for all $A \in \text{attr}(R)$. The runs of identical hyperplanes partition the n_R-dimensional space of $\sigma(R)$ into at most $(2K+1)^{n_R}$ many n_R-dimensional grid boxes. Each dimension of a grid box is formed by one of the (at most) $2K+1$ runs from one attribute. By construction, these grid boxes form a partition of the n_R-dimensional space as each grid box is a distinct combination of equivalence classes for the attributes and the orderings returned by Algorithm 3 cover all the values in $\text{dom}(A)$. Figure 4b demonstrates the grid boxes implied by the equivalence classes in the orderings of a relation. In the figure, there are two equivalence classes for attribute A and three for B, dividing the relation into 6 grid boxes.

We argue that each grid box is completely full of either gaps or tuples. Let $t \in R$, let g be the grid box containing t, and let t' be another point in g. Two points t_1, t_2 are *adjacent* if for some attribute A, $t_1.A$ and $t_2.A$ are adjacent in σ_A. Consider moving from t to t' through any sequence of adjacent points in g. When we pass through a point we are moving from one A-hyperplane to an identical A-hyperplane for some attribute A. Thus every point along this path must also be a tuple in R. A similar argument for gaps implies that every point in a grid box that contains one gap must also be a gap. Since the grid boxes partition the domain of R, constructing one box for each gap grid box results in a box cover $\mathcal{B}_R$ for R. Since there are at most $(2K+1)^{n_R}$ grid boxes, $|\mathcal{B}_R| \leq (2K+1)^{n_R}$. We can construct such a box cover for each $R \in \mathcal{R}$ to obtain a box cover for $\sigma(Q)$ of size $\sum_{R \in \mathcal{R}} (2K+1)^{n_R} \leq m(2K+1)^r = \widetilde{O}(K^r)$, completing the proof of ADORA's approximation ratio. ◀

Appendix E shows that our analysis of ADORA's approximation factor is asymptotically tight by defining a family of queries over binary relations which have orderings with box covers of size K, whereas the orderings that ADORA returns require $\Omega(K^2)$ boxes.

5.3 TetrisReordered

We next combine ADORA, GAMB, and Tetris in a new join algorithm we call *TetrisReordered* to obtain new beyond worst-case optimal results for join queries. Algorithm 4 presents the pseudocode of TetrisReordered. Corollary 14 immediately follows from Theorems 4 and 13 from this paper, and Theorems 4.9 and 4.11 from reference [1].

Algorithm 3 OrderAttr(Q, A): Groups equivalence classes for A into consecutive runs.

$\phi :=$ any attribute ordering of $\mathcal{A}$ which places A first
$\mathcal{S} := \{R \in \mathcal{R} : A \in \text{attr}(R)\}$, $D := \bigcup_{R \in \mathcal{S}} \pi_A(R)$, $\mathcal{T} := \emptyset$
for $R \in \mathcal{S}$ **do**
    Sort R lexicographically according to ϕ
for $a \in D$ **do**
    $\mathcal{T}[a] := []$
    for $R \in \mathcal{S}$ in a fixed order **do**
        $\mathcal{T}[a]$.append$(H(R, A, a))$
Sort D by ordering a_i and a_j according to the lexicographic order of $\mathcal{T}[a_i]$ and $\mathcal{T}[a_j]$
return $\sigma_A = D$ (append $a \notin D$ to σ_A in arbitrary order)

▶ **Corollary 14.** *Let $Q = (\mathcal{R}, \mathcal{A})$ be a join query. Let w be the treewidth of Q, $n = |\mathcal{A}|$, and r the maximum arity of a relation in $\mathcal{R}$. Let σ^* be an optimal solution to $\texttt{DomOr}_{\texttt{BoxMinB}}$ on Q and let $K = K_\square(\sigma^*(Q))$. TetrisReordered computes Q in $\widetilde{O}(N + K^{r(w+1)} + Z)$ time by using Tetris-Reloaded or in $\widetilde{O}(N + K^{rn/2} + Z)$ time by using Tetris-LoadBalanced as a subroutine.*

Algorithm 4 TetrisReordered(Q).

$\sigma :=$ ADORA(Q) (Algorithm 2)
for $R \in \mathcal{R}$ **do**
    $\mathcal{B}_R :=$ GAMB$(\sigma(Q))$ (Algorithm 1)
return σ^{-1}(Tetris$(\mathcal{B} = \{\mathcal{B}_R\}_{R \in \mathcal{R}})$)

We next show that there are infinite families of queries for which these bounds are arbitrarily smaller than prior bounds stated for Tetris in reference [1]. In fact, given an arbitrary query Q with any number of output tuples and any certificate size, and a default domain ordering σ, we can generate families of queries for which TetrisReordered is unboundedly faster than Tetris on σ. Our method can be seen as a generalization of the "checkerboard" example in Figure 2.[6] Take an arbitrary query $Q = (\mathcal{R}, \mathcal{A})$ with N input tuples and Z output tuples. Recall that the minimum certificate size for Q under its default ordering is denoted $C_\square(Q)$. Let σ_{ADR} be the ordering ADORA generates on Q and let U be the corresponding upper bound on $K_\square(\sigma_{ADR}(Q))$ provided by Theorem 13. Recall that U depends on r, the maximum arity of a relation in $\mathcal{R}$, and is an upper bound on the minimum box cover and certificate size for $\sigma_{ADR}(Q)$. We generate a family of queries Q_p from Q for $p = 1, 2, ...$, whose minimum certificate size (under σ) increases to $2^{rp}C_\square(Q)$ but the upper bound provided by ADORA according to Theorem 13 remains at U.

Let $\mathcal{A}_p$ be the attribute set obtained from $\mathcal{A}$ by adding, for each $A \in \mathcal{A}$, an additional p bits as a prefix to the d bits of A. For every relation $R \in \mathcal{R}$, construct a relation R_p with attr$(R_p) \subseteq \mathcal{A}_p$ corresponding to attr(R). For each $t \in R$, add the following tuples to R_p:

$$\{\langle p_A t.A\rangle_{A \in \text{attr}(R)} : p_A \in \{0,1\}^p \ \forall A \in \text{attr}(R)\}$$

[6] The example in Figure 2 is a simplified version of the one used to prove Lemma J.1 in reference [1], which shows that *general resolutions*, which are logical operations on two DNF clauses, are more powerful than geometric resolutions, which are constrained to contiguous geometric boxes.

The p bits added to each attribute do not affect the structure of the query, since these bits vary over all possible valuations for each tuple from the original query. For each attribute A, these bits effectively create 2^p "copies" of each A-hyperplane. This increases the size of the query's input, output, and box certificate under the default ordering. The query $Q_p = (\mathcal{R}_p, \mathcal{A}_p)$ has input size $2^{rp}N$, output size $2^{np}Z$, and minimum box certificate size $2^{rp}C_\square(Q)$, where r is the maximum arity of a relation in $\mathcal{R}$ and $n = |\mathcal{A}|$. However, this construction does not affect the number of equivalence classes on any dimension. Instead, it increases the size of each equivalence class by a factor of 2^p. To see this, consider two values of an attribute $A \in \mathcal{A}$, a_1 and a_2, that were in the same equivalence class in Q. That is, they had the same A-hyperplanes for every relation $R \in \mathcal{R}$ such that $A \in \text{attr}(R)$. After adding the p bits, there will be 2^p "copies" of a_1 and a_2, one for each p-bit prefix that was appended to tuples that contained a_1 and a_2. Each copy will have the same (but larger) A-hyperplane. The number of equivalence classes on each attribute will remain the same, so the bound of Theorem 13 on $K_\square(\sigma_{ADR}(Q))$ will remain at U. As p increases, the performance gap between TetrisReordered and prior versions of Tetris becomes arbitrarily large.

6 Open Problems

The problems defined in this paper are ripe for further study. Beyond NP-hardness, we know little about $\texttt{DomOr}_{\texttt{BoxMinC}}$. Even if the domain ordering is fixed, we do not know of a way to approximate the minimum certificate size that is asymptotically faster than computing the join. Appendix B.3 in reference [1] describes how to use a variant of Minesweeper to compute a certificate. In Appendix G of the online version of this paper [4], we show a variant of Tetris can do the same. These approaches effectively compute the join to compute a certificate. A difficult aspect of this problem is that a certificate is a box cover for the output relation using boxes from the input relations. However, since the output tuples are not known a priori, the exact space that needs to be covered is not known at domain reordering time. It is also not known a priori which input tuples are part of the output and therefore which gap boxes from the input relations are part of the certificate.

Similarly, little is known about the following problems: (1) determining whether a specific gap box is in the minimum-size certificate under a fixed domain ordering; (2) verifying that a given domain ordering induces a box cover or certificate of minimum size; and (3) variants of $\texttt{DomOr}_{\texttt{BoxMinB}}$ and $\texttt{DomOr}_{\texttt{BoxMinC}}$ under additional assumptions about the structure of the input relations. As an example, Appendix F shows that if the input relations are fully semi-join reduced (a problem known to be hard for cyclic queries [7]), so all input tuples are part of the output and all gap boxes are relevant to the certificate, then the minimum box cover and certificate sizes are within an $\widetilde{O}(1)$ factor of one another. In this special case, ADORA approximates $\texttt{DomOr}_{\texttt{BoxMinC}}$.

Each of these open problems can be defined over general or dyadic boxes, creating two related but distinct problems. Sufficiently strong hardness of approximation results for either version of a problem would imply the difference is negligible, since the solutions would be within an $\widetilde{O}(1)$-factor of one another by Lemma 3. Theorem 9 shows that the general box versions of $\texttt{DomOr}_{\texttt{BoxMinB}}$ and $\texttt{DomOr}_{\texttt{BoxMinC}}$ are NP-hard, but this theorem does not imply a hardness result about the dyadic versions of these problems. Proving a hardness of approximation result is a direction for future study. Until such a result is known, the distinction between general and dyadic boxes is crucial to studying these problems.

Developing an ADORA-like preprocessing algorithm that provides similar results but is query-independent, or has a better approximation ratio, is also a direction for future research.

ADORA runs in $\widetilde{O}(N)$ time, so its query-dependence precludes the possibility of a sub-linear time join algorithm using ADORA. The domain ordering must be computed from scratch for each different query, even if the relations in the database have not changed. Sharing some of this computation between different queries would improve on our results.

7 Conclusions

For queries with fixed domain orderings, we established a $\widetilde{O}(N)$-time algorithm GAMB to create a single globally good box cover index which is guaranteed to contain a certificate at most a $\widetilde{O}(1)$ factor larger than the minimum size certificate for any box cover. We then studied $\texttt{DomOr}_{\texttt{BoxMinB}}$, the problem of finding a domain ordering that yields the smallest possible box cover size for a given query Q. We proved that $\texttt{DomOr}_{\texttt{BoxMinB}}$ is NP-hard and presented a $\widetilde{O}(N)$-time approximation algorithm ADORA that computes an ordering which yields a box cover of size $\widetilde{O}(K^r)$, where K is the minimum box cover size under any ordering and r is the maximum arity of any relation in Q. We combined ADORA, GAMB, and Tetris in an algorithm we call TetrisReordered and stated new beyond worst-case optimal runtimes for join processing in Corollary 14. TetrisReordered can improve the known performance bounds of prior versions of Tetris (on any fixed ordering) on infinite families of queries.

Our work leaves several interesting problems open as discussed in Section 6. Our results are limited to the problems $\texttt{DomOr}_{\texttt{BoxMinB}}$ and $\texttt{DomOr}_{\texttt{BoxMinC}}$, and there are several interesting variants of these problems for which little is known.

References

1. Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond, April 2014. `arXiv:1404.0703`.
2. Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems*, 41(4), December 2016.
3. Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2016.
4. Kaleb Alway, Eric Blais, and Semih Salihoglu. Box covers and domain orderings for beyond worst-case join processing, September 2019. `arXiv:1909.12102`.
5. Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4), 2013.
6. Piotr Berman and Bhaskar DasGupta. Complexities of efficient solutions of rectilinear polygon cover problems. *Algorithmica*, 17(4), April 1997.
7. Philip A. Bernstein and Dah ming W. Chiu. Using Semi-joins to Solve Relational Queries. *Journal of the ACM*, 28(1), January 1981.
8. Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3), December 1976.
9. John Horton Conway and Neil James Alexander Sloane. *Sphere Packings, Lattices and Groups*, volume 290. Springer Science & Business Media, 2013.
10. Joseph C. Culberson and Robert A. Reckhow. Covering polygons is hard. *Journal of Algorithms*, 17(1), July 1994.
11. Rina Dechter and Judea Pearl. Tree-clustering schemes for constraint-processing. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*, 1988.
12. Deborah S. Franzblau. Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles. *SIAM Journal on Discrete Mathematics*, 2(3), August 1989.

13 Deborah S. Franzblau and Daniel J. Kleitman. An algorithm for covering polygons with rectangles. *Information and Control*, 63(3), December 1984.

14 Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *Journal of the ACM*, 59(3), June 2012.

15 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1), October 2014.

16 Joachim Gudmundsson and Christos Levcopoulos. Close approximations of minimum rectangular coverings. *Journal of combinatorial optimization*, 3(4), December 1999.

17 Salim Haddadi. A note on the NP-hardness of the consecutive block minimization problem. *International Transactions in Operational Research*, 9, November 2002.

18 Manas R. Joglekar and Christopher M. Ré. It's all a matter of degree: Using degree information to optimize multiway joins. In *19th International Conference on Database Theory*, 2016.

19 Lawrence T. Kou. Polynomial complete consecutive information retrieval problems. *SIAM Journal on Computing*, 6(1), March 1977.

20 V.S. Anil Kumar and H. Ramesh. Covering rectilinear polygons with axis-parallel rectangles. *SIAM Journal on Computing*, 32(6), October 2003.

21 Christos Levcopoulos and Joachim Gudmundsson. Approximation algorithms for covering polygons with squares and similar problems. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, 1997.

22 Anna Lubiw. Doubly lexical orderings of matrices. *SIAM Journal on Computing*, 16(5), October 1987.

23 Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2018.

24 Hung Q. Ngo, Dung T. Nguyen, Christopher Ré, and Atri Rudra. Beyond worst-case analysis for joins with Minesweeper. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2014.

25 Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM*, 65(3), March 2018.

26 Hung Q Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: New developments in the theory of join algorithms. *ACM SIGMOD Record*, 42(4), February 2014.

27 Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems*, 40(1), March 2015.

28 Gábor Fejes Tóth and Wlodzimierz Kuperberg. A survey of recent results in the theory of packing and covering. In *New Trends in Discrete and Computational Geometry*, pages 251–279. Springer Berlin Heidelberg, 1993.

29 Todd L. Veldhuizen. Leapfrog triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of the 17th International Conference on Database Theory*, 2014.

30 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases*, 1981.

A Example with $\omega(N)$ Maximal General Gap Boxes

In Section 4, GAMB generates all maximal dyadic gap boxes in $\widetilde{O}(N)$ time. The use of dyadic boxes instead of general boxes in GAMB is necessary, because there are relations for which the number of maximal general gap boxes is asymptotically greater than the number of tuples in the relation. Our construction generalizes the example in Figure 15 in Appendix B.3 of reference [1]. Let N be an even number, A and B be attributes over domains of size N, and R_N be the following relation.

$$R_N(A,B) = \{\langle i, N/2-i-1\rangle : 0 \leq i < N/2\} \cup \{\langle N/2+i, N-i-1\rangle : 0 \leq i < N/2\}$$

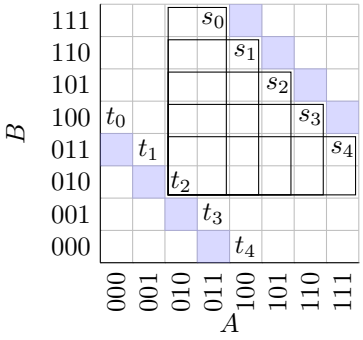


Figure 5 Example of a relation $R_8(A, B)$ with $\omega(N)$ maximal general gap boxes.

Algorithm 5 InsertMDBCI(R, B, t): Update B after t is inserted to R.

```
1: for each b ∈ B such that t ∈ b do
2:     B := B \ {b}
3:     for each b' such that t ∈ b' ⊆ b do
4:         for A ∈ attr(R) such that |b.A| < d do
5:             Let b'' be the box when one bit is added to b'.A such that t ∉ b''
6:             B := B ∪ {b''}
```

Consider the following sets of tuples which are *not* in R_N.

$T_N = \{t_i = \langle i, N/2{-}i\rangle : 0 \le i \le N/2\}$

$S_N = \{s_i = \langle N/2{+}i{-}1, N{-}i{-}1\rangle : 0 \le i \le N/2\}$

Figure 5 depicts R_8, T_8, and S_8. In this diagram, a set of 5 maximal general gap boxes with bottom left corners at t_2 is depicted. The top right corners of these boxes are the 5 tuples in S_8. In fact, for each $0 \le i \le 4$, there are 4 or 5 maximal general gap boxes in R_8 with their bottom left corner at t_i. This property generalizes from R_8 to any value of N. For each $0 \le i \le N/2$, there are at least $N/2$ maximal general gap boxes in R_N with their bottom left corner at t_i. Since there are $N/2 + 1$ tuples in T_N, the total number of maximal general gap boxes in R_N is at least $(N/2 + 1)(N/2) = \Theta(N^2) = \omega(N)$.

B Incremental Maintenance of a Maximal Box Cover Index

Theorem 4 states that running GAMB on a relation R produces a set of dyadic gap boxes containing all maximal dyadic gap boxes of R. In this section, we refer to such a set as a *maximal dyadic box cover index (MDBCI)* for R. An MDBCI can be thought of as an index for R that can be computed once and used in any query over R. Traditional database indexes are useful because they are easy to incrementally maintain when tuples are added to or removed from a relation, so the index does not need to be computed from scratch every time the relation is modified. The following results show that efficient incremental maintenance is also possible with MDBCIs. The proofs of Theorems 15, 16, and 17 can be found in Appendix B of the online version of this paper [4].

▶ **Theorem 15.** *Suppose that R is a relation with MDBCI B. Let $t \notin R$. Let $R' = R \cup \{t\}$ and let B' be the result of running Algorithm 5 with R', B and t as input. Then B' is an MDBCI for R'.*

Algorithm 6 DeleteMDBCI(R, B, t): Update B after t is deleted from R.

```
1: for every dyadic box b such that t ∈ b do
2:     addb := True
3:     for each b′ such that t ∈ b′ ⊂ b do
4:         for A ∈ attr(R) such that b′.A ≠ ∗ do
5:             Let b″ be the box when the last bit of b′.A is flipped
6:             if there is no box in B that contains b″ then addb := False
7:     if addb then
8:         B := B ∪ {b}
```

▶ **Theorem 16.** *Suppose that R is a relation with maximal dyadic box cover index B. Let $t \in R$. Let $R' = R \setminus \{t\}$ and let B' be the result of running Algorithm 6 with R', B and t as input. Then B' is an MDBCI for R'.*

▶ **Theorem 17.** *Algorithms 5 and 6 run in $\widetilde{O}(1)$ time.*

C Proof that $\texttt{DomOr}_{\texttt{BoxMinB}}$ is NP-hard

This section contains proofs of the 6 transformation steps we used in the proof of Theorem 9.

C.1 Proof of Step 1

▷ Claim. *Every $r_{i,j}$ row can be made adjacent to some equivalent $r_{k,\ell}$ row.*

Let $r_1 := r_{i,j}$ be a row which is not adjacent to any equivalent row. Let $r_2 := r_{k,\ell}$ be any row equivalent to r_1 (at least one such row exists because we duplicate each row of M when constructing M'). Since r_1 is not adjacent to any equivalent row and there are an even number of rows equivalent to r_1, there must be some run E of rows equivalent to r_1 with odd length. If E has length 1, we assume r_2 is the one row in E, and therefore r_2 is not adjacent to any equivalent row. If E has length at least 3, we assume r_2 is the second row in E, and therefore r_2 is not adjacent to $p_{k,\ell}$. Let $p_1 := p_{i,j}$ and let $p_2 := p_{k,\ell}$. Let c_{p1} be the column where p_1 has a 1-cell, and let c_{p2} be the column where p_2 has a 1-cell. Let c_1 and c_2 be the columns where r_1 and r_2 both have 1-cells. Let $b_1 \in B$ be the box covering the padding column in r_1 with greatest width. Let $b_2 \in B$ be the box covering the padding column in r_2 with greatest width. Let $b_3 \in B$ be the box covering the padding column in p_1. Let $b_4 \in B$ be the box covering the padding column in p_2. Our approach will be to remove the rows r_1, r_2, p_1, and p_2 from M', then insert them in the order (p_1, r_1, r_2, p_2) at the bottom of M'. In this order, the 1-cells of these rows can be covered by 4 boxes, regardless of the column ordering. A box of width 1 and height 2 can be used to cover the two 1-cells in each of the columns in $\{c_1, c_2, c_{p1}, c_{p2}\}$. To show that this modification does not increase the number of boxes in B, it suffices to show that there are at least 4 boxes which can be removed from B when we remove these 4 rows from M'. We split our analysis into four cases.

1. $b_1 \neq b_3$ and $b_2 \neq b_4$. In this case, all of $\{b_1, b_2, b_3, b_4\}$ are distinct and all 4 of these boxes are removed when we remove the rows r_1, r_2, p_1, p_2.
2. $b_1 \neq b_3$ and $b_2 = b_4$. Since $b_2 = b_4$, r_2 is adjacent to p_2. By our previous assumptions about r_2, r_2 is not adjacent to any equivalent row. W.l.o.g., assume that p_2 is directly below r_2. Let r_3 be the row directly above r_2. r_3 is not equivalent to r_2, so there exists

a box b_5 covering at least one of c_1 or c_2 in r_2 with height 1, since it cannot extend vertically to either p_2 or r_3. b_5 is not equal to b_2, because b_2 has height 2. $\{b_1, b_2, b_3, b_5\}$ is a set of 4 distinct boxes which are removed when we remove the rows $\{r_1, r_2, p_1, p_2\}$.

3. $b_1 = b_3$ and $b_2 \neq b_4$. Since $b_1 = b_3$, r_1 is adjacent to p_1. Suppose w.l.o.g. that p_1 is directly above r_1. Let r_3 be the row directly below r_1. Since r_1 is not adjacent to any equivalent rows, r_3 is not equivalent to r_1. Therefore, there is a box $b_6 \in B$ covering at least one of c_1 or c_2 in r_1 which has height 1, since it cannot extend vertically to either p_1 or r_3. b_6 is not equal to b_1, since b_1 has height 2. Now the set of boxes $\{b_1, b_2, b_4, b_6\}$ is a set of 4 distinct boxes which are removed when we remove the rows $\{r_1, r_2, p_1, p_2\}$.
4. $b_1 = b_3$ and $b_2 = b_4$. This case can be proven by combining the arguments from the previous two cases. Since $b_2 = b_4$, we can define the box b_5 exactly as in case 2. Since $b_1 = b_3$, we can define the box b_6 exactly as in case 3. Then, $\{b_1, b_2, b_5, b_6\}$ is a set of 4 distinct boxes which are removed from B when we remove the rows $\{r_1, r_2, p_1, p_2\}$.

C.2 Proof of Step 2

▷ Claim. *Every run of equivalent $r_{i,j}$ rows can be made to have even length.*

Let E_1 be a run of equivalent $r_{i,j}$ rows of odd length. By the claim of step 1, E_1 has length ≥ 3. Let r_1 be the second row in E_1. Since E_1 has odd length and there are an even number of rows equivalent to r_1, there exists another run E_2 of rows equivalent to r_1 with odd length. E_2 also has length ≥ 3. Let c_{p1} be the column which has a 1-cell only in r_1 and its corresponding padding row. Let $b \in B$ be the box which covers c_{p1} in r_1. Since r_1 is not adjacent to its padding row, b has height 1. If we remove r_1 from M', b can be removed. By inserting r_1 directly below the first row in E_2, a unit box can be used to cover c_{p1} in r_1. Let r_2 be the first row in E_2. Let c_1 and c_2 be the two columns of M' where r_1 and r_2 share 1-cells. To cover these two 1-cells in r_1, we can extend vertically the boxes covering c_1 and c_2 in r_2. We may assume these boxes can be extended vertically, because at most two of the rows in E_2 have their c_1 (or c_2) cell covered by a box which stretches horizontally from a padding column. That is, there is *some* row in E_2 where the box covering the c_1 (or c_2) cell can be extended vertically to cover the c_1 (or c_2) cell of r_1. Hence, this transformation can be made without increasing the size of B. After this, both E_1 and E_2 have even length. Continue this process until every run of equivalent $r_{i,j}$ rows have even length.

C.3 Proof of Step 3

▷ Claim. *Every run of equivalent $r_{i,j}$ rows can be made to have length 2.*

Let E be a run of equivalent $r_{i,j}$ rows of even length greater than 2. So E has a length of at least 4. Let r_1 be the second row in E and let r_2 be the third row in E. Since E has length at least 4, neither r_1 nor r_2 are adjacent to their respective padding rows, p_1 and p_2. We claim the boxes covering the padding columns in r_1 and r_2 have width 1. We split our analysis into two cases. Let c_1 and c_2 be the two columns where r_1 and r_2 both have 1-cells.

1. c_1 and c_2 are adjacent. At most 2 of the rows in E have their padding columns adjacent to (c_1, c_2) on either side. This means there is some row r_3 in E where the box b covering c_1 and c_2 does not also cover its padding column. b can be extended vertically to cover c_1 and c_2 in all rows of E. Any boxes covering padding columns for rows in E can be replaced with boxes of width 1, and all of the 1-cells in the rows of E remain covered.
2. c_1 and c_2 are not adjacent. At most 2 rows in E have their padding columns adjacent to c_1 on either side. This means there is some row r_3 in E where the box b covering c_1 does

not also cover its padding column. b can be extended vertically to cover c_1 in all rows of E. The same argument applies for c_2. Any boxes covering padding columns for rows in E can be replaced with boxes of width 1, and all 1-cells in the rows of E remain covered.

Now, removing p_1 and p_2 removes two boxes from B, since unit boxes must be covering the single 1-cells in p_1 and p_2. Inserting (p_1, p_2) in order in between r_1 and r_2, we can cover the 1-cells in (c_{p1}, p_1) and (c_{p2}, p_2) by extending vertically the width 1 boxes covering (c_{p1}, r_1) and (c_{p2}, r_2). This splits any boxes which vertically streched from r_1 to r_2 into two. There were at most two such boxes, so the total number of boxes in B does not increase. Now E is split into two distinct runs of equivalent rows, one of length 2 and one of length $|E| - 2$. This process can be repeated until all runs have length exactly 2.

C.4 Proof of Step 4

▷ Claim. *The padding rows $p_{i,j}$ can be made adjacent to their matching $r_{i,j}$ rows.*

Let $r_1 := r_{i,j}$ be a row which is not adjacent to its padding row $p_1 := p_{i,j}$. By the claim of step 3, we know r_1 is adjacent to exactly one row, r_2, that is equivalent to r_1. Let p_2 be the padding row matching r_2. Let c_1 and c_2 be the columns where r_1 and r_2 share 1-cells. Let c_{p1} be the column which has 1-cells only in r_1 and p_1. Let c_{p2} be the column which has 1-cells only in r_2 and p_2. Let b_1 be the box which covers the 1-cell in row r_1 and column c_{p1} of greatest width. Let b_2 be the box which covers the 1-cell in row r_2 and column c_{p2} of greatest width. Let b_3 be the box which covers the 1-cell in p_1. Let b_4 be the box which covers the 1-cell in p_2. We split our analysis into two cases.

1. r_2 is adjacent to p_2. In this case, similar to our argument in step 1, there exists a box $b_5 \in B$ with height 1 which covers c_1 or c_2 (or both) in r_2. By removing the rows $\{r_1, r_2, p_1, p_2\}$, the 4 distinct boxes $\{b_1, b_2, b_3, b_5\}$ are all removed from B. By inserting the rows (p_1, r_1, r_2, p_2) in order at the bottom of the matrix, we can cover their 1-cells with at most 4 boxes, so the total number of boxes in B does not increase.
2. r_2 is not adjacent to p_2. In this case, r_1 is not adjacent to p_1 and r_2 is not adjacent to p_2, so $\{b_1, b_2, b_3, b_4\}$ are 4 distinct boxes in B which are removed if we remove rows $\{r_1, r_2, p_1, p_2\}$. By inserting (p_1, r_1, r_2, p_2) in order at the bottom of the matrix, we can cover their 1-cells with at most 4 boxes, so the size of B does not increase.

We can repeat this process until all $r_{i,j}$ rows are adjacent to their matching $p_{i,j}$ rows.

C.5 Proof of Step 5

▷ Claim. *The row order σ'_r can be made to exactly match the default row order of M'.*

By the claims of steps 3 and 4, all of the rows are now divided into separate 4-row units containing a run of two equivalent $r_{i,j}$ rows surrounded by their two matching padding rows. There are no boxes in B which can stretch vertically across two or more of these separate units, because there are no two $p_{i,j}$ rows which share a 1-cell. Thus, we are free to reorder these units arbitrarily. Order the units so that for all i, the i-th unit contains two $r_{i,j}$ rows which correspond to the i-th row of the original matrix M. The resulting row order σ'_r is then equal to the default row ordering of M', modulo any equivalent rows which are swapped from their default positions. Since equivalent rows are equal up to reordering the columns of M', there exists an ordering on the columns of M' that transforms $\sigma'_r(M')$ back to the original matrix M'. In other words, the row ordering σ'_r is now equivalent to the default row ordering of M' up to a relabelling of the rows. This is sufficient for our purposes, since we can relabel the rows accordingly and move on to modifying the column ordering only.

C.6 Proof of Step 6

▷ Claim. *The column order σ'_c can be made to exactly match the default column order of M' on the last $2n$ columns.*

For each padding row $p_{i,j}$, the box b covering the single 1-cell in $p_{i,j}$ has width 1. By step 4, each padding row is adjacent to its corresponding $r_{i,j}$ row. This means b extends vertically to also cover the only other 1-cell in its column. Therefore, by moving this column to the right side of the matrix, we do not increase the total number of boxes in B. Once all of these padding columns have been moved to the right, the boxes covering all of their 1-cells all have width 1. Thus, we can reorder them to exactly match the last $2n$ columns in the default column ordering of M' without modifying any boxes in B.

D ADORA's Runtime Analysis

ADORA calls Algorithm 3 n, so $\widetilde{O}(1)$, times. In Algorithm 3, the sorting of m relations according to ϕ on line 4 takes $\widetilde{O}(N)$ time. The for-loop beginning on line 5 iterates over each domain value $a \in D$ and each $R \in \mathcal{S}$ and appends $H(R, A, a)$ to $\mathcal{T}[a]$. Since R was sorted lexicographically according to ϕ, which places A as the first relation, all tuples with the same A-value are now consecutive in R. Therefore, with a single linear pass through R, we can compute all of the hyperplanes $H(R, A, a)$. We do this for each relation, so the runtime is bounded by $O(mN) = \widetilde{O}(N)$. For the final sorting of D on line 9 observe that the total size of the array $\mathcal{T}$, summed over all domain values a, is at most N. Thus, we are sorting an array of arrays where the total amount of data is of size $\widetilde{O}(N)$, which can be done in $\widetilde{O}(N)$ time (e.g., with a merge-sort algorithm that merges two sorted sub-arrays in $\widetilde{O}(N)$ time), completing the proof that ADORA's runtime is $\widetilde{O}(N)$ as claimed in Theorem 13.

E The ADORA Approximation Bound Is Tight

Theorem 13 proved that ADORA produces a domain ordering σ for Q such that $K_\square(\sigma(Q)) = \widetilde{O}(K^r)$, where K is the minimum box cover size under any domain ordering and r is the maximum arity of a relation in Q. We will show that this bound is tight by presenting a class of 2D relations R_d for which ADORA returns a domain ordering σ such that $K_\square(\sigma(R_d)) = \Omega(K^2)$, where K is the minimum box cover size for R_d under any ordering. For any integer $d > 0$, let $R_d(A, B)$ be the relation over 2 d-bit attributes A and B given by

$$R_d(A, B) = \{\langle 0a, 0b\rangle : a, b \in \{0, 1\}^{d-1}, a \neq b\} \cup \{\langle 1a, 1b\rangle : a, b \in \{0, 1\}^{d-1}, a \neq b\}$$

As an example, R_3 is illustrated in Appendix E of the online version of this paper [4]. The default ordering of R_d has a minimum box cover of size $K = 2^d + 1$. However, there is a bad ordering σ_d such that $\sigma_d(R_d)$ has a minimum box cover size of $\Omega(2^{2d}) = \Omega(K^2)$. The key observation about this example is that no rows or columns in R_d are equal, so ADORA may return σ_d as a solution. Since R_d has arity 2, the bound of Theorem 13 is tight in this case.

F Approximating $\texttt{DomOr}_{\texttt{BoxMinC}}$ On Fully Semi-join Reduced Queries

This section serves to illustrate that if the input relations of a query Q are fully semi-join reduced, so we know a priori that all of the input tuples contribute to the query's output, then $\texttt{DomOr}_{\texttt{BoxMinC}}$ can be approximated with ADORA. We use the term *"dangling" (input) tuple as follows. Assume the domain ordering σ is fixed. Given a query $Q = (\mathcal{R}, \mathcal{A})$ (under*

σ) and a relation $R \in \mathcal{R}$, the tuple $t \in R$ is a dangling tuple if there is no tuple t' in the output of Q such that $\pi_{\text{attr}(R)}(t') = t$. Q is said to be fully semi-join reduced [7] if there are no dangling tuples in any of the relations in $\mathcal{R}$. The problem of fully semi-join reducing a query by removing all of the dangling tuples is known to be hard for cyclic queries [7]. Proposition 18 (proven in Appendix F of the online version of this paper [4]) shows that an oracle which computes the full semi-join reduction of a query Q would allow us to bridge the gap between minimizing the box cover size and certificate size for Q.

▶ **Proposition 18.** *Q be a fully semi-join reduced query under σ. Let $K_\square(Q)$ be the size of the minimum box cover for Q under σ. Let $C_\square(Q)$ be the size of the minimum certificate of Q under σ. Then $K_\square(Q) = \widetilde{\Theta}(C_\square(Q))$.*

A Purely Regular Approach to Non-Regular Core Spanners

Markus L. Schmid
Humboldt-Universität zu Berlin, Germany

Nicole Schweikardt
Humboldt-Universität zu Berlin, Germany

Abstract

The regular spanners (characterised by vset-automata) are closed under the algebraic operations of union, join and projection, and have desirable algorithmic properties. The core spanners (introduced by Fagin, Kimelfeld, Reiss, and Vansummeren (PODS 2013, JACM 2015) as a formalisation of the core functionality of the query language AQL used in IBM's SystemT) additionally need string equality selections and it has been shown by Freydenberger and Holldack (ICDT 2016, Theory of Computing Systems 2018) that this leads to high complexity and even undecidability of the typical problems in static analysis and query evaluation. We propose an alternative approach to core spanners: by incorporating the string-equality selections directly into the regular language that represents the underlying regular spanner (instead of treating it as an algebraic operation on the table extracted by the regular spanner), we obtain a fragment of core spanners that, while having slightly weaker expressive power than the full class of core spanners, arguably still covers the intuitive applications of string equality selections for information extraction and has much better upper complexity bounds of the typical problems in static analysis and query evaluation.

2012 ACM Subject Classification Information systems → Information retrieval; Theory of computation → Automata extensions; Theory of computation → Regular languages; Theory of computation → Design and analysis of algorithms; Theory of computation → Database query languages (principles)

Keywords and phrases Document spanners, regular expressions with backreferences

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.4

Related Version *Full Version*: `https://arxiv.org/abs/2010.13442`

Funding *Markus L. Schmid*: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 416776735 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 416776735).
Nicole Schweikardt: Partially supported by the ANR project EQUUS ANR-19-CE48-0019; funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 431183758 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 431183758).

Acknowledgements We wish to thank the anonymous reviewers for their valuable feedback that improved the readability of this paper.

1 Introduction

The information extraction framework of *document spanners* has been introduced by Fagin, Kimelfeld, Reiss, and Vansummeren [4] as a formalisation of the query language AQL, which is used in IBM's information extraction engine SystemT. A document spanner performs information extraction by mapping a *document*, formalised as a word w over a finite alphabet Σ, to a relation over so-called *spans* of w, which are intervals $[i, j\rangle$ with $0 \leqslant i < j \leqslant |w| + 1$.

The document spanners (or simply *spanners*, for short) introduced in [4] follow a two-stage approach: *Primitive* spanners extract relations directly from the input document, which are then further manipulated by using some relational algebra. As primitive spanners, [4] introduces *vset-automata* and *regex-formulas*, which are variants of nondeterministic finite automata and regular expressions, respectively, that can use meta-symbols ${}^{\mathsf{x}}\triangleright$ and $\triangleleft^{\mathsf{x}}$, where x is a *variable* from a set $\mathcal{X}$ of variables, in order to bind those variables to start and end positions of spans, therefore extracting an $|\mathcal{X}|$-ary span-relation, or a table with columns labelled by the variables $\mathcal{X}$. For example, $\alpha = ({}^{\mathsf{x}}\triangleright\, (\mathtt{a} \vee \mathtt{b})^* \triangleleft^{\mathsf{x}}) \cdot ({}^{\mathsf{y}}\triangleright\, (\mathtt{a}^* \vee \mathtt{b}^*) \triangleleft^{\mathsf{y}}) \mathtt{c}^*$ is a regex-formula and it describes a spanner $[\![\alpha]\!]$ by considering for a given word w all possibilities of how w can be generated by α and for each such generation of w, the variables x and y extract the spans that correspond to those subwords of w that are generated by the subexpressions ${}^{\mathsf{x}}\triangleright\, (\mathtt{a} \vee \mathtt{b})^* \triangleleft^{\mathsf{x}}$ and ${}^{\mathsf{y}}\triangleright\, (\mathtt{a}^* \vee \mathtt{b}^*) \triangleleft^{\mathsf{y}}$, respectively. For example, on input $w = \mathtt{abaac}$, we have $[\![\alpha]\!](w) = \{([1,3\rangle, [3,5\rangle), ([1,4\rangle, [4,5\rangle), ([1,5\rangle, [5,5\rangle)\}$, since α can generate ${}^{\mathsf{x}}\triangleright\, \mathtt{ab} \triangleleft^{\mathsf{x}}\, {}^{\mathsf{y}}\triangleright\, \mathtt{aa} \triangleleft^{\mathsf{y}} \mathtt{c}$, ${}^{\mathsf{x}}\triangleright\, \mathtt{aba} \triangleleft^{\mathsf{x}}\, {}^{\mathsf{y}}\triangleright\, \mathtt{a} \triangleleft^{\mathsf{y}} \mathtt{c}$ and ${}^{\mathsf{x}}\triangleright\, \mathtt{abaa} \triangleleft^{\mathsf{x}}\, {}^{\mathsf{y}}\triangleright\, \triangleleft^{\mathsf{y}} \mathtt{c}$. The vset-automata follow the same principle, but take the form of nondeterministic finite automata. Since these primitive spanners are based on formal language description mechanisms, they are also called *regular spanners* and, for the sake of presentation, we denote this class of regular spanners by $\mathsf{reg}\text{-}\mathfrak{S}$ for the remainder of this introduction (there are different ways of characterising *regular spanners* and also different semantics (see [12, 4]); these aspects shall be discussed in more detail below).

The considered algebraic operations are union $\cup$, natural join $\bowtie$, projection π (with the obvious meaning) and string-equality selection $\varsigma^{=}_{\mathcal{Z}}$, which is a unary operator parameterised by a set $\mathcal{Z} \subseteq \mathcal{X}$ of variables, and it selects exactly those rows of the table for which all spans of columns $\mathcal{Z}$ refer to (potentially different occurrences of) the same subwords of w.

The *core spanners* (capturing the *core* of SystemT's query language AQL) introduced in [4] are defined as $\mathsf{reg}\text{-}\mathfrak{S}^{\{\cup,\bowtie,\pi,\varsigma^{=}\}}$, i. e., the closure of regular spanners under the operations $\cup$, $\bowtie$, π and $\varsigma^{=}$ (these relational operations are interpreted as operations on spanners in the natural way). A central result of [4] is that the operations $\cup$, $\bowtie$ and π can be directly incorporated into the regular spanners, i. e., $\mathsf{reg}\text{-}\mathfrak{S}^{\{\cup,\bowtie,\pi\}} = \mathsf{reg}\text{-}\mathfrak{S}$. This is due to the fact that regular spanners are represented by finite automata and therefore the closure properties for regular languages carry over to regular spanners by similar automaton constructions. This also holds in the case of so-called *schemaless semantics* (see [12]). However, as soon as we also consider the operator of string-equality selection, the picture changes considerably.

In terms of expressive power, it can be easily seen that not all core spanners are regular spanners, simply because for all regular spanners S the language $\{w \in \Sigma^* \mid S(w) \neq \emptyset\}$ is regular, which is not necessarily the case for core spanners. As shown in [4], we can nevertheless represent any core spanner $S \in \mathsf{reg}\text{-}\mathfrak{S}^{\{\cup,\bowtie,\pi,\varsigma^{=}\}}$ in the form $\pi_{\mathcal{Y}} \varsigma^{=}_{\mathcal{Z}_1} \varsigma^{=}_{\mathcal{Z}_2} \dots \varsigma^{=}_{\mathcal{Z}_k}(S')$ for a regular spanner S' (this is called the *core-simplification lemma* in [4]).

Regular spanners have excellent algorithmic properties: enumerating $S(w)$ can be done with linear preprocessing and constant delay, even if the spanner is given as vset-automaton (see [1, 6]), while spanner containment or inclusion can be decided efficiently if the spanner is represented by a certain deterministic vset-automaton (see [3]). However, in terms of complexity, we have to pay a substantial price for adding string-equality selections to regular spanners. It has been shown in [8] that for core spanners the typical problems of query evaluation and static analysis are NP- or PSpace-hard, or even undecidable (see Table 1).

The results from [8] identify features that are, from an intuitive point of view, sources of complexity for core spanners. Thus, the question arises whether tractability can be achieved by restricting core spanners respectively. We shall illustrate this with some examples.

Table 1 Comparison of decision problems of regular spanners, core spanners and refl-spanners. A formal definition of the problems can be found in Section 5. In the case of regular spanners and refl-spanners, the input spanner is represented by an NFA M. The abbreviation "str. ref." means *strongly reference extracting*, a restriction for refl-spanners to be formally defined in Section 5. The authors are not aware of a (non-trivial) upper bound for ModelChecking for regular spanners (note that since regular spanners are covered by refl-spanners, the upper bound for refl-spanners applies).

	Problem	Regular sp.	Refl-sp.	Core sp. [8]
Evaluation problems	ModelChecking	?	$\text{poly}(\lvert M\rvert)(\lvert w\rvert + (2\lvert\mathcal{X}\rvert)!)$ [T. 5.1]	NP-c
	NonEmptiness	$O(\lvert M\rvert\lvert w\rvert)$	NP-c [T. 5.2]	NP-h
Static analysis problems	Satisfiability	$O(\lvert M\rvert)$	$O(\lvert M\rvert)$ [T. 5.3]	PSpace-c
	Containment	PSpace-c [12]	ExpSpace (for str. ref.) [T. 5.10]	undec.
	Equivalence	PSpace-c [12]	ExpSpace (for str. ref.) [T. 5.10]	undec.
	Hierarchicality	$O(\lvert M\rvert\lvert\mathcal{X}\rvert^3)$	$O(\lvert M\rvert\lvert\mathcal{X}\rvert^3)$ [T. 5.3]	PSpace-c

Consider a regex formula $\alpha = {}^{\mathsf{x}_1\triangleright}\Sigma^*\triangleleft^{\mathsf{x}_1}\,{}^{\mathsf{x}_2\triangleright}\Sigma^*\triangleleft^{\mathsf{x}_2}\dots{}^{\mathsf{x}_n\triangleright}\Sigma^*\triangleleft^{\mathsf{x}_n}$. Then checking, for some $\mathcal{Z}_1, \mathcal{Z}_2, \dots, \mathcal{Z}_k \subseteq \{\mathsf{x}_1, \mathsf{x}_2, \dots, \mathsf{x}_n\}$, whether the empty tuple is in $(\pi_\emptyset \varsigma^{=}_{\mathcal{Z}_1}\varsigma^{=}_{\mathcal{Z}_2}\dots\varsigma^{=}_{\mathcal{Z}_k}([\![\alpha]\!]))(w)$, is identical to checking whether w can be factorised into n factors such that for each $\mathcal{Z}_i$ all factors that correspond to the variables in $\mathcal{Z}_i$ are the same. This is the pattern matching problem with variables (or the membership problem for pattern languages), a well-known NP-complete problem (see, e. g., [11]). However, checking for a (non-empty) span-tuple t whether it is in $(\varsigma^{=}_{\mathcal{Z}_1}\varsigma^{=}_{\mathcal{Z}_2}\dots\varsigma^{=}_{\mathcal{Z}_k}([\![\alpha]\!]))(w)$ can be easily done in polynomial time, since the task of checking the existence of a suitable factorisation boils down to the task of evaluating a factorisation that is implicitly given by t. Hence, instead of blaming the string-equality selections for intractability, we could as well blame the projection operator. Can we achieve tractability by restricting projections instead of string-equality selections?

Another feature that yields intractability is that we can use string-equality selections in order to concisely express the intersection non-emptiness of regular languages (a well-known PSpace-complete problem). For example, let $r_1, r_2, \dots, r_n$ be some regular expressions, and let $\alpha = {}^{\mathsf{x}_1\triangleright}r_1\triangleleft^{\mathsf{x}_1}\,{}^{\mathsf{x}_2\triangleright}r_2\triangleleft^{\mathsf{x}_2}\dots{}^{\mathsf{x}_n\triangleright}r_n\triangleleft^{\mathsf{x}_n}$. Then there is a word w with $(\varsigma^{=}_{\{\mathsf{x}_1,\mathsf{x}_2,\dots,\mathsf{x}_n\}}([\![\alpha]\!]))(w) \neq \emptyset$ if and only if $\bigcap_{i=1}^{n}\mathcal{L}(r_i) \neq \emptyset$. So string-equality selections do not only check whether the same subword has several occurrences, but also, as a "side-effect", check membership of this repeated subword in the intersection of several regular languages. Can we achieve tractability by somehow limiting the power of string-equality selections to the former task?

A third observation is that by using string-equality selections on *overlapping* spans, we can use core spanners to express rather complex word-combinatorial properties. In fact, we can even express word equations as core spanners (see [8, Proposition 3.7, Example 3.8, Theorem 3.13] for details). Can we achieve tractability by requiring all variables that are subject to string-equality selections to extract only pairwise non-overlapping spans?

1.1 Our Contribution

We introduce *refl-spanners* (based on *regular ref-languages*), a new formalism for spanners that properly extends regular spanners, describes a large class of core spanners, and has better upper complexity bounds than core spanners. Moreover, the formalism is purely based on regular language description mechanisms. The main idea is a paradigm shift in the two-stage approach of core spanners: instead of extracting a span-relation with a regular spanner and then applying string-equality selections on it, we handle string-equality selections directly with the finite automaton (or regular expression) that describes the regular spanner. However,

checking the equality of unbounded factors in strings is a task that, in most formalisms, can be considered highly "non-regular" (the well-known *copy-language* $\{ww \mid w \in \Sigma^*\}$ is a textbook example for demonstrating the limits of regular languages in this regard). We deal with this obstacle by representing the factors that are subject to string-equality selections as *variables* in the regular language. For example, while $L = \{\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \geqslant 0\}$ is non-regular, the language $L' = \{ {}^{\mathtt{x}}\triangleright\, \mathtt{a}^n \,\triangleleft^{\mathtt{x}} \mathtt{b}\mathtt{x} \mid n \geqslant 0\}$ can be interpreted as a regular description of L by means of meta-symbols ${}^{\mathtt{x}}\triangleright$ and $\triangleleft^{\mathtt{x}}$ to *capture* a factor, and a meta-symbol $\mathtt{x}$ to *copy* or *reference* the captured factor. In particular, all words of L can be easily obtained from the words of L' by simply replacing the occurrence of $\mathtt{x}$ with the factor it refers to. As long as core spanners use string equality selections in a not too complicated way, this simple formalism seems also to be suited for describing core spanners, e. g., the core spanner $\pi_{\{\mathtt{x},\mathtt{y}\}}\varsigma^{=}_{\{\mathtt{x},\mathtt{x}'\}}\varsigma^{=}_{\{\mathtt{y},\mathtt{y}'\}}([\![\alpha]\!])$ with $\alpha = {}^{\mathtt{x}}\triangleright\mathtt{a}^*\mathtt{b}^{\mathtt{y}}\triangleright\mathtt{c}\triangleleft^{\mathtt{x}}\mathtt{b}^*\,{}^{\mathtt{x}'}\triangleright\mathtt{a}^*\mathtt{b}\mathtt{c}\triangleleft^{\mathtt{x}'}\,\triangleleft^{\mathtt{y}}\,{}^{\mathtt{y}'}\triangleright\mathtt{c}\mathtt{b}^*\mathtt{a}^*\mathtt{b}\mathtt{c}\triangleleft^{\mathtt{y}'}$ could be represented as $[\![{}^{\mathtt{x}}\triangleright\mathtt{a}^*\mathtt{b}^{\mathtt{y}}\triangleright\mathtt{c}\triangleleft^{\mathtt{x}}\mathtt{b}^*\mathtt{x}\triangleleft^{\mathtt{y}}\mathtt{y}]\!]$.

The class of refl-spanners can now informally be described as the class of all spanners that can be represented by a regular language over the alphabet $\Sigma \cup \mathcal{X} \cup \{{}^{\mathtt{x}}\triangleright, \triangleleft^{\mathtt{x}} \mid \mathtt{x} \in \mathcal{X}\}$ that has the additional property that the meta-symbols $\mathcal{X} \cup \{{}^{\mathtt{x}}\triangleright, \triangleleft^{\mathtt{x}} \mid \mathtt{x} \in \mathcal{X}\}$ are "well-behaved" in the sense that each word describes a valid span-tuple (one of our main conceptional contributions is to formalise this idea in a sound way).

The refl-spanner formalism automatically avoids exactly the features of core spanners that we claimed above to be sources of complexity. More precisely, refl-spanners cannot project out variables, which means that they cannot describe the task of checking the existence of some complicated factorisation. Furthermore, it can be easily seen that in the refl-spanner formalism, we cannot describe intersection non-emptiness of regular languages in a concise way, as is possible by core spanners. Finally, we can only have overlaps with respect to the spans captured by ${}^{\mathtt{x}}\triangleright \ldots \triangleleft^{\mathtt{x}}$, but all references $\mathtt{x}$ represent pairwise non-overlapping factors, which immediately shows that we cannot express word equations as core spanners can. This indicates that refl-spanners are restricted in terms of expressive power, but it also gives hope that for refl-spanners we can achieve better upper complexity bounds of the typical decision problems compared to core spanners, and, in fact, this is the case (see Table 1).

It is obvious that not all core spanners can be represented as refl-spanners, but we can nevertheless show that a surprisingly large class of core spanners can be handled by the refl-spanner formalism. Recall that the core simplification lemma from [4] states that in every core spanner $S \in \mathsf{reg}\text{-}\mathfrak{S}^{\{\cup,\pi,\bowtie,\varsigma^{=}\}}$, we can "push" all applications of $\cup$ and $\bowtie$ into the automaton that represents the regular spanner, leaving us with an expression $\pi_{\mathcal{Y}}\varsigma^{=}_{\mathcal{Z}_1}\varsigma^{=}_{\mathcal{Z}_2}\ldots\varsigma^{=}_{\mathcal{Z}_k}(M)$ for an automaton M that represents a regular spanner. We can show that if the string-equality selections $\varsigma^{=}_{\mathcal{Z}_1}\varsigma^{=}_{\mathcal{Z}_2}\ldots\varsigma^{=}_{\mathcal{Z}_k}$ apply to a set of variables that never capture overlapping spans, then we can further "push" even all string-equality selections into M, turning it into a representation of a refl-spanner that "almost" describes S: in order to get S, we only have to merge certain columns into a single one by creating the fusion of the corresponding spans.

1.2 Related Work

Spanners have recently received a lot of attention [4, 9, 15, 1, 12, 6, 16, 7, 8, 10, 13]. However, as it seems, most of the recent progress on document spanners concerns regular spanners. For example, it has recently been shown that results of regular spanners can be enumerated with linear preprocessing and constant delay [1, 6], the paper [12] is concerned with different semantics of regular spanners and their expressive power, and [15] investigates the evaluation of algebraic expressions over regular spanners.

Papers that are concerned with string-equality selection are [8] in which many negative results for core spanner evaluation are shown, [7] which, by presenting a logic that exactly covers core spanners, answers question on the expressive power of core spanners, [16] that

shows that datalog over regular spanners covers the whole class of core spanners, and [9] which investigates conjunctive queries on top of regular spanners and, among mostly negative results, also contains the positive result that such queries with equality-selections can be evaluated efficiently if the number of string equalities is bounded by a constant. The paper [10] investigates the dynamic descriptive complexity of regular spanners and core spanners. While all these papers contribute deep insights with respect to document spanners, positive algorithmic results for the original core spanners from [4] seem scarce and the huge gap in terms of tractability between regular and core spanners seems insufficiently bridged by tractable fragments of core spanners.

A rather recent paper that also deals with non-regular document spanners is [14]. However, the non-regular aspect of [14] does not consist in string-equality selections, but rather that spanners are represented by context-free language descriptors (in particular grammars) instead of regular ones.

1.3 Organisation

The rest of the paper is structured as follows. Section 2 fixes the basic notation concerning spanners and lifts the core-simplification lemma of [4] to the schemaless case. In Section 3, we develop a simple declarative approach to spanners by establishing a natural one-to-one correspondence between spanners and so-called subword-marked languages. In Section 4 we extend the concept of subword-marked languages in order to describe spanners with string-equality selections which we call refl-spanners. Section 5 is devoted to the complexity of evaluation and static analysis problems for refl-spanners. Section 6 studies the expressive power of refl-spanners. We conclude the paper in Section 7. Due to space constraints, most proof details are omitted, although we give proof sketches for some results; detailed proofs can be found in the preliminary full version of this paper [18].

2 Preliminaries

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $[n] = \{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$. For a (partial) mapping $f : X \to Y$, we write $f(x) = \perp$ for some $x \in X$ to denote that $f(x)$ is not defined; we also set $\mathrm{dom}(f) = \{x \mid f(x) \neq \perp\}$. By $\mathcal{P}(A)$ we denote the power set of a set A, and A^+ denotes the set of non-empty words over A, and $A^* = A^+ \cup \{\varepsilon\}$, where ε is the empty word. For a word $w \in A^*$, $|w|$ denotes its length (in particular, $|\,\varepsilon\,| = 0$), and for every $b \in A$, $|w|_b$ denotes the number of occurrences of b in w. Let A and B be alphabets with $B \subseteq A$, and let $w \in A^*$. Then $\mathsf{e}_B : A \to A \cup \{\varepsilon\}$ is a mapping with $\mathsf{e}_B(b) = \varepsilon$ if $b \in B$ and $\mathsf{e}_B(b) = b$ if $b \in A \setminus B$; we also use e_B to denote the natural extension of e_B to the morphism $A^* \to A^*$. Technically, e_B depends on the alphabet A, but whenever we use $\mathsf{e}_B(w)$ we always assume that $\mathsf{e}_B : A \to A \cup \{\varepsilon\}$ for some alphabet A with $w \in A^*$.

2.1 Regular Language Descriptors

For an alphabet Σ, the set RE_Σ of *regular expressions* (*over* Σ) is defined as usual: every $a \in \Sigma \cup \{\varepsilon\}$ is in RE_Σ with $\mathcal{L}(a) = \{a\}$, and, for $r, s \in \mathsf{RE}_\Sigma$, $(r \cdot s), (r \vee s), (r)^+ \in \mathsf{RE}_\Sigma$ with $\mathcal{L}((r \cdot s)) = \mathcal{L}(r) \cdot \mathcal{L}(s)$, $\mathcal{L}((r \vee s)) = \mathcal{L}(r) \cup \mathcal{L}(s)$, $\mathcal{L}((r)^+) = (\mathcal{L}(r))^+$. For $r \in \mathsf{RE}_\Sigma$, we use r^* as a shorthand form for $(r)^+ \vee \varepsilon$, and we usually omit the operator "$\cdot$", i. e., we use juxtaposition. For the sake of readability, we often omit parentheses, if this does not cause ambiguities.

A *nondeterministic finite automaton* (NFA for short) is a tuple $M = (Q, \Sigma, \delta, q_0, F)$ with a set Q of states, a finite alphabet Σ, a start state q_0, a set F of accepting states and a transition function $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$. We also interpret NFA as directed, edge-labelled graphs in the obvious way. A word $w \in \Sigma^*$ is accepted by M if there is a path from q_0 to some $q_f \in F$ that is labelled by w; $\mathcal{L}(M)$ is the accepted language, i. e., the set of all accepted words. The size $|M|$ of an NFA is measured in $|Q| + |\delta|$. However, we will mostly consider NFA with constant out-degree, which means that $|M| = \mathrm{O}(|Q|)$. For a language descriptor D (e. g., an NFA or a regular expression), we denote by $\mathcal{L}(D)$ the language defined by D. The class of languages described by NFA or regular expressions is the class of regular languages, denoted by reg-$\mathfrak{L}$.

2.2 Spans and Spanners

For a word $w \in \Sigma^*$ and for every $i, j \in [|w|+1]$ with $i \leqslant j$, $[i, j\rangle$ is a *span of* w and its *value*, denoted by $w[i, j\rangle$, is the substring of w from symbol i to symbol $j-1$. In particular, $w[i, i\rangle = \varepsilon$ (this is called *empty span*) and $w[1, |w|+1\rangle = w$. By $\mathsf{Spans}(w)$, we denote the set of spans of w, and by Spans we denote the set of spans for any word (elements from Spans shall simply be called *spans*). A span $[i, j\rangle$ can also be interpreted as the set $\{i, i+1, \ldots, j-1\}$ and therefore we can use set-theoretical notions for spans. Two spans $s = [i, j\rangle$ and $s' = [i', j'\rangle$ are *equal* if $s = s'$, they are *disjoint* if $j \leqslant i'$ or $j' \leqslant i$ and they are *quasi-disjoint* if they are equal or disjoint. Note that s and s' being disjoint is sufficient, but not necessary for $s \cap s' = \emptyset$, e. g., $[3, 6\rangle$ and $[5, 5\rangle$ are not disjoint, but $[3, 6\rangle \cap [5, 5\rangle = \emptyset$.

For a finite set of variables $\mathcal{X}$, an $(\mathcal{X}, w)$-*tuple* (also simply called *span-tuple*) is a partial function $\mathcal{X} \to \mathsf{Spans}(w)$, and a $(\mathcal{X}, w)$-*relation* is a set of $(\mathcal{X}, w)$-tuples. For simplicity, we usually denote $(\mathcal{X}, w)$-tuples in tuple-notation, for which we assume an order on $\mathcal{X}$ and use the symbol "$\perp$" for undefined variables, e. g., $([1, 5\rangle, \perp, [5, 7\rangle)$ describes a $(\{\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3\}, w)$-tuple that maps x_1 to $[1, 5\rangle$, x_3 to $[5, 7\rangle$, and is undefined for x_2.

An $(\mathcal{X}, w)$-tuple t is *functional* if it is a total function, t is *hierarchical* if, for every $\mathsf{x}, \mathsf{y} \in \mathrm{dom}(t)$, $t(\mathsf{x}) \subseteq t(\mathsf{y})$ or $t(\mathsf{y}) \subseteq t(\mathsf{x})$ or $t(\mathsf{x}) \cap t(\mathsf{y}) = \emptyset$, and t is *quasi-disjoint* if, for every $\mathsf{x}, \mathsf{y} \in \mathrm{dom}(t)$, $t(\mathsf{x})$ and $t(\mathsf{y})$ are quasi-disjoint. An $(\mathcal{X}, w)$-relation is *functional*, *hierarchical* or *quasi-disjoint*, if all its elements are functional, hierarchical or quasi-disjoint, respectively.

A *spanner* (*over terminal alphabet* Σ *and variables* $\mathcal{X}$) is a function that maps every $w \in \Sigma^*$ to an $(\mathcal{X}, w)$-relation (note that the empty relation $\emptyset$ is also a valid image of a spanner). Since the dependency on the word w is often negligible, we also use the term $\mathcal{X}$-*tuple* or $\mathcal{X}$-*relation* to denote an $(\mathcal{X}, w)$-tuple or $(\mathcal{X}, w)$-relation, respectively.

▶ **Example 2.1.** Let $\Sigma = \{\mathsf{a}, \mathsf{b}\}$ and let $\mathcal{X} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$. Then the function S that maps words $w \in \Sigma^*$ to the $(\mathcal{X}, w)$-relation $\{([1, i\rangle, [i, i+1\rangle, [i+1, |w|+1\rangle) \mid 1 \leqslant i < |w|, w[i, i+1\rangle = \mathsf{b}\}$ is a spanner. For example, $S(\mathsf{ababbab}) = \{t_1, t_2, t_3, t_4\}$ with $t_1 = ([1, 2\rangle, [2, 3\rangle, [3, 8\rangle)$, $t_2 = ([1, 4\rangle, [4, 5\rangle, [5, 8\rangle)$, $t_3 = ([1, 5\rangle, [5, 6\rangle, [6, 8\rangle)$ and $t_4 = ([1, 7\rangle, [7, 8\rangle, [8, 8\rangle)$.

Let S_1 and S_2 be spanners over Σ and $\mathcal{X}$. Then S_1 and S_2 are said to be *equal* if, for every $w \in \Sigma^*$, $S_1(w) = S_2(w)$ (this coincides with the usual equality of functions and shall also be denoted by $S_1 = S_2$). We say that S_2 *contains* S_1, written as $S_1 \subseteq S_2$, if, for every $w \in \Sigma^*$, $S_1(w) \subseteq S_2(w)$. A spanner S over Σ and $\mathcal{X}$ is *functional*, *hierarchical* or *quasi-disjoint* if, for every w, $S(w)$ is functional, hierarchical or quasi-disjoint, respectively. Note that, for span-tuples, span-relations and spanners, quasi-disjointness implies hierarchicality.

Next, we define operations on spanners. The *union* $S_1 \cup S_2$ of two spanners S_1 and S_2 over Σ and $\mathcal{X}$ is defined via $(S_1 \cup S_2)(w) = S_1(w) \cup S_2(w)$ for all $w \in \Sigma^*$.

The *natural join* $S_1 \bowtie S_2$ is defined via $(S_1 \bowtie S_2)(w) = S_1(w) \bowtie S_2(w)$ for all $w \in \Sigma^*$. Here, for two $(\mathcal{X}, w)$-relations R_1, R_2 we let $R_1 \bowtie R_2 = \{t_1 \bowtie t_2 \mid t_1 \in R_1, t_2 \in R_2, t_1 \sim t_2\}$. Two $(\mathcal{X}, w)$-tuples t_1 and t_2 are *compatible* (denoted by $t_1 \sim t_2$) if $t_1(\mathsf{x}) = t_2(\mathsf{x})$ for every $\mathsf{x} \in \mathrm{dom}(t_1) \cap \mathrm{dom}(t_2)$, and for compatible $(\mathcal{X}, w)$-tuples t_1 and t_2, the $(\mathcal{X}, w)$-tuple $t_1 \bowtie t_2$ is defined by $(t_1 \bowtie t_2)(\mathsf{x}) = t_i(\mathsf{x})$ if $\mathsf{x} \in \mathrm{dom}(t_i)$ for $i \in \{1, 2\}$.

The *projection* $\pi_{\mathcal{Y}}(S_1)$ for a set $\mathcal{Y} \subseteq \mathcal{X}$ is defined by letting $(\pi_{\mathcal{Y}}(S_1))(w) = \{t_{|\mathcal{Y}} \ : \ t \in S_1(w)\}$, where $t_{|\mathcal{Y}}$ is the restriction of t to domain $\mathrm{dom}(t) \cap \mathcal{Y}$.

The *string-equality selection* $\varsigma^{=}_{\mathcal{Y}}(S_1)$ for a set $\mathcal{Y} \subseteq \mathcal{X}$ is defined by letting $(\varsigma^{=}_{\mathcal{Y}}(S_1))(w)$ contain all $t \in S_1(w)$ such that, for every $\mathsf{x}, \mathsf{y} \in \mathcal{Y} \cap \mathrm{dom}(t)$, if $t(\mathsf{x}) = [i, j\rangle$ and $t(\mathsf{y}) = [i', j'\rangle$, then $w[i, j\rangle = w[i', j'\rangle$. Further below, we will discuss why we only require $w[i, j\rangle = w[i', j'\rangle$ for $\mathsf{x}, \mathsf{y} \in \mathcal{Y} \cap \mathrm{dom}(t)$ instead of $\mathsf{x}, \mathsf{y} \in \mathcal{Y}$.

For convenience, we omit the parentheses if we apply sequences of unary operations of spanners, e. g., we write $\pi_{\mathcal{Z}} \varsigma^{=}_{\mathcal{Y}_1} \varsigma^{=}_{\mathcal{Z}_1}(S)$ instead of $\pi_{\mathcal{Z}}(\varsigma^{=}_{\mathcal{Y}_1}(\varsigma^{=}_{\mathcal{Z}_1}(S)))$. For any $E = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_\ell\} \subseteq \mathcal{P}(\mathcal{X})$, we also write $\varsigma^{=}_{E}(S)$ instead of $\varsigma^{=}_{\mathcal{Y}_1} \varsigma^{=}_{\mathcal{Y}_2} \ldots \varsigma^{=}_{\mathcal{Y}_\ell}(S)$, and in this case we will call $\varsigma^{=}_{E}$ a *generalised* string-equality selection, or also just string-equality selection if it is clear from the context that $E \subseteq \mathcal{P}(\mathcal{X})$. For generalised string-equality selections $\varsigma^{=}_{E}$, we will always assume that all sets of E are pairwise disjoint. For a class $\mathfrak{S}$ of spanners and set P of spanner operations, $\mathfrak{S}^P$ (or $\mathfrak{S}^p$ if $P = \{p\}$) denotes the closure of $\mathfrak{S}$ under the operations from P.

Whenever formulating complexity bounds, we consider the terminal alphabet Σ to be constant, but we always explicitly state any dependency on $|\mathcal{X}|$.

2.3 Regular Spanners and Core Spanners

In [4], the class of *regular spanners*, denoted by $\mathsf{reg}\text{-}\mathfrak{S}$, is defined as the class of spanners represented by *vset-automata*, and the class of *core spanners* is defined as $\mathsf{core}\text{-}\mathfrak{S} = [\![\mathsf{RGX}]\!]^{\{\cup, \pi, \bowtie, \varsigma^{=}\}}$, where RGX is the class of so-called *regex-formulas* (we refer to [4] for a formal definition of vset-automata and regex-formulas). A crucial result from [4] is the *core-simplification lemma*: every $S \in \mathsf{core}\text{-}\mathfrak{S}$ can be represented as $\pi_{\mathcal{Y}} \varsigma^{=}_{E}(S')$, where S' is a regular spanner. The setting in [4] uses a *function semantics* for spanners, i. e., $(\mathcal{X}, w)$-tuples are always functional. In our definitions above, we allow variables in span-tuples and spanners to be undefined, i. e., we use partial mappings as introduced in [12], and in the terminology of [15], we consider the *schemaless* semantics.

In [12], it is shown that the classical framework for *regular spanners* with function semantics introduced in [4] can be extended to the schemaless case, i. e., vset-automata and regex-formula are extended to the case of schemaless semantics, and it is shown that the basic results still hold (e. g., vset-automata are equally powerful as regex-formulas (or vstack-automata) equipped with union, natural join and projection). However, the string-equality selection operator – which turns regular spanners into the more powerful core spanners – is not treated in [12]. Our definition of the string-equality selection operator given above extends the definition from [4] from the functional to the schemaless case by interpreting $\varsigma^{=}_{\mathcal{Y}}$ to apply only to those variables from $\mathcal{Y}$ that are in the domain of the span-tuple. This way of treating undefined variables is natural and also corresponds to how the join operator is extended to the schemaless case in [12]. Due to [12], we can also in the schemaless case define $\mathsf{reg}\text{-}\mathfrak{S}$ as the class of spanners defined by vset-automata (with schemaless semantics), and we can also define the class of core spanners with schemaless semantics as $\mathsf{core}\text{-}\mathfrak{S} = [\![\mathsf{RGX}]\!]^{\{\cup, \pi, \bowtie, \varsigma^{=}\}}$. However, to the knowledge of the authors, the core-simplification lemma from [4] has so far not been extended to the schemaless semantics. Since we wish to apply the core-simplification lemma in the context of our results (for schemaless semantics), and since this seems to be

a worthwhile task in its own right, we show that the core-simplification lemma from [4] holds verbatim for the schemaless case. For those parts of the proof's argument that are not concerned with string-equality selections, we heavily rely on the results from [12].

▶ **Lemma 2.2** (Core Simplification Lemma). *For every* $S \in \mathsf{core}\text{-}\mathfrak{S}$ *over* $\mathcal{X}$ *there are* $S' \in \mathsf{reg}\text{-}\mathfrak{S}$, $\mathcal{Y} \subseteq \mathcal{X}$ *and* $E \subseteq \mathcal{P}(\mathcal{X})$ *such that* $S = \pi_{\mathcal{Y}} \varsigma^{=}_{E}(S')$.

3 A Declarative Approach to Spanners

In this section, we develop a simple declarative approach to spanners by establishing a natural one-to-one correspondence between spanners over Σ and so-called *subword-marked languages* over Σ.[1] This approach conveniently allows to investigate or define non-algorithmic properties of spanners completely independently from any machine model or other description mechanisms (e. g., types of regular expressions, automata, etc.), while at the same time we can use the existing algorithmic toolbox for formal languages whenever required (instead of inventing special-purpose variants of automata or regular expressions to this end).

In particular, this declarative approach is rather versatile and provides some modularity in the sense that we could replace "regular languages" by any kind of language class (e. g., (subclasses of) context-free languages, context-sensitive languages, etc.) to directly obtain (i. e., without any need to adopt our definitions) a formally sound class of document spanners and also have the full technical machinery that exists for this language class at our disposal. Note that the idea of using non-regular languages from the Chomsky hierarchy to define more powerful classes of document spanners has been recently used in [14].

In the context of this paper, however, the main benefit is that this approach provides a suitable angle to treat string-equality selections in a regular way.

3.1 Subword-Marked Words

For any set $\mathcal{X}$ of variables, we shall use the set $\Gamma_{\mathcal{X}} = \{{}^{\mathsf{x}}\triangleright, \triangleleft^{\mathsf{x}} \mid \mathsf{x} \in \mathcal{X}\}$ as an alphabet of meta-symbols. In particular, for every $\mathsf{x} \in \mathcal{X}$, we interpret the pair of symbols ${}^{\mathsf{x}}\triangleright$ and $\triangleleft^{\mathsf{x}}$ as a pair of opening and closing parentheses.

▶ **Definition 3.1** (Subword-Marked Words). *A* subword-marked *word (*over terminal alphabet Σ and variables $\mathcal{X}$*) is a word* $w \in (\Sigma \cup \Gamma_{\mathcal{X}})^*$ *such that, for every* $\mathsf{x} \in \mathcal{X}$, $\mathsf{e}_{\Sigma \cup \Gamma_{\mathcal{X}\setminus\{\mathsf{x}\}}}(w) \in \{\varepsilon, {}^{\mathsf{x}}\triangleright\, \triangleleft^{\mathsf{x}}\}$. *A subword-marked word is* functional, *if* $|w|_{{}^{\mathsf{x}}\triangleright} = 1$ *for every* $\mathsf{x} \in \mathcal{X}$. *For a subword-marked word* w *over* Σ *and* $\mathcal{X}$, *we set* $\mathfrak{e}(w) = \mathsf{e}_{\Gamma_{\mathcal{X}}}(w)$.

A subword-marked word w can be interpreted as a word over Σ, i. e., the word $\mathfrak{e}(w)$, in which some subwords are marked by means of the parentheses ${}^{\mathsf{x}}\triangleright$ and $\triangleleft^{\mathsf{x}}$. In this way, it represents an $(\mathcal{X}, \mathfrak{e}(w))$-tuple, i. e., every $\mathsf{x} \in \mathcal{X}$ is mapped to $[i, j\rangle \in \mathsf{Spans}(\mathfrak{e}(w))$, where $w = w_1\, {}^{\mathsf{x}}\triangleright\, w_2\, \triangleleft^{\mathsf{x}}\, w_3$ with $i = |\mathfrak{e}(w_1)| + 1$ and $j = |\mathfrak{e}(w_1 w_2)| + 1$. In the following, the $(\mathcal{X}, \mathfrak{e}(w))$-tuple defined by a subword-marked word w is denoted by $\mathsf{st}(w)$. We note that $\mathsf{st}(w)$ is a total function if and only if w is functional. Moreover, we say that a subword-marked word w is *hierarchical* or *quasi-disjoint*, if $\mathsf{st}(w)$ is hierarchical or quasi-disjoint, respectively.

[1] In the literature on spanners, subword-marked words have previously been used as a tool to define the semantics of regex-formulas or vset-automata (see, e. g., [3, 7, 9, 10]). However, in these papers, the term *ref-word* is used instead of subword-marked word, which is a bit of a misnomer due to the following reasons. Ref-words have originally been used in [17] (in a different context) as words that contain *references* to some of their subwords, which are explicitly marked. In the context of spanners, only ref-words with marked subwords, but *without* any references have been used so far. Since in this work we wish to use ref-words in the sense of [17], i. e., with actual references, but also the variants without references, we introduce the term subword-marked word for the latter.

▶ **Example 3.2.** Let $\mathcal{X} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$ and $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Then $\mathsf{x}\triangleright\, \mathtt{aa} \,\triangleleft^{\mathsf{x}}\, \mathtt{ab}\, \mathsf{y}\triangleright\, \mathsf{z}\triangleright\, \mathtt{ca} \,\triangleleft^{\mathsf{z}}\, \mathtt{a}\triangleleft^{\mathsf{y}}$ is a functional and hierarchical subword-marked word. The subword-marked word $u = \mathtt{b}\, \mathsf{x}\triangleright\, \mathtt{a}\, \mathsf{y}\triangleright\, \mathtt{aba}\, \mathsf{z}\triangleright\, \mathtt{a} \triangleleft^{\mathsf{z}} \mathtt{c} \triangleleft^{\mathsf{x}} \mathtt{ab} \triangleleft^{\mathsf{y}} \mathtt{c}$ is functional, but not hierarchical, while $v = \mathsf{x}\triangleright\, \mathtt{a}\, \mathsf{y}\triangleright\, \mathtt{ba} \triangleleft^{\mathsf{y}} \mathtt{cab} \triangleleft^{\mathsf{x}} \mathtt{caa}$ is a non-functional, but hierarchical subword-marked word. Moreover, $\mathsf{st}(u) = ([2,8\rangle, [3,10\rangle, [6,7\rangle)$ and $\mathsf{st}(v) = ([1,7\rangle, [2,4\rangle, \bot)$. On the other hand, neither $\mathsf{x}\triangleright\, \mathtt{aa} \,\triangleleft^{\mathsf{x}}\, \mathtt{ab}\, \mathsf{y}\triangleright\, \mathsf{x}\triangleright\, \mathtt{ca} \,\triangleleft^{\mathsf{x}}\, \mathtt{a}\triangleleft^{\mathsf{y}}$ nor $\mathsf{x}\triangleright\, \mathtt{a}\, \mathsf{y}\triangleright\, \mathtt{ba} \triangleleft^{\mathsf{x}} \mathtt{c}$ are valid subword-marked words.

3.2 Subword-Marked Languages and Spanners

A set L of subword-marked words (over Σ and $\mathcal{X}$) is a *subword-marked language* (*over* Σ *and* $\mathcal{X}$); a subword-marked language L is called *functional*, *hierarchical* or *quasi-disjoint* if all $w \in L$ are functional, hierarchical or quasi-disjoint, respectively. Since every subword-marked word w over Σ and $\mathcal{X}$ describes a $(\mathcal{X}, \mathfrak{e}(w))$-tuple, subword-marked languages can be interpreted as spanners as follows.

▶ **Definition 3.3.** *Let L be a subword-marked language (over Σ and $\mathcal{X}$). Then the spanner $[\![L]\!]$ (over Σ and $\mathcal{X}$) is defined as follows: for every $w \in \Sigma^*$, $[\![L]\!](w) = \{\mathsf{st}(v) \mid v \in L, \mathfrak{e}(v) = w\}$. For a class $\mathcal{L}$ of subword-marked languages, we set $[\![\mathcal{L}]\!] = \{[\![L]\!] \mid L \in \mathcal{L}\}$.*

▶ **Example 3.4.** Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and $\mathcal{X} = \{\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3\}$. Let $\alpha = \mathsf{x}_1\triangleright\, (\mathtt{a} \vee \mathtt{b})^* \triangleleft^{\mathsf{x}_1}\, \mathsf{x}_2\triangleright\, \mathtt{b} \triangleleft^{\mathsf{x}_2}\, \mathsf{x}_3\triangleright\, (\mathtt{a} \vee \mathtt{b})^* \triangleleft^{\mathsf{x}_3}$ be a regular expression over the alphabet $\Sigma \cup \Gamma_{\mathcal{X}}$. We can note that $\mathcal{L}(\alpha)$ is a subword-marked language (over terminal alphabet Σ and variables $\mathcal{X}$) and therefore $[\![\mathcal{L}(\alpha)]\!]$ is a spanner over $\mathcal{X}$. In fact, $[\![\mathcal{L}(\alpha)]\!]$ is exactly the spanner described by the function S in Example 2.1.

In this way, every subword-marked language L over Σ and $\mathcal{X}$ describes a spanner $[\![L]\!]$ over Σ and $\mathcal{X}$, and since it is also easy to transform any $(\mathcal{X}, w)$-tuple t into a subword-marked word v with $\mathfrak{e}(v) = w$ and $\mathsf{st}(v) = t$, also every spanner S over Σ and $\mathcal{X}$ can be represented by a subword-marked language over Σ and $\mathcal{X}$. Moreover, for a subword-marked language L over Σ and $\mathcal{X}$, $[\![L]\!]$ is a functional, hierarchical or quasi-disjoint spanner if and only if L is functional, hierarchical or quasi-disjoint, respectively. This justifies that we can use the concepts of spanners (over Σ and $\mathcal{X}$) and the concept of subword-marked languages (over Σ and $\mathcal{X}$) completely interchangeably. By considering only *regular* subword-marked languages, we automatically obtain the class of regular spanners (usually defined as the class of spanners that can be described by vset-automata [4, 12]). More formally, let $\mathsf{reg\text{-}swm\text{-}}\mathfrak{L}_{\Sigma,\mathcal{X}}$ be the class of regular subword-marked languages over Σ and $\mathcal{X}$ and let $\mathsf{reg\text{-}swm\text{-}}\mathfrak{L} = \bigcup_{\Sigma,\mathcal{X}} \mathsf{reg\text{-}swm\text{-}}\mathfrak{L}_{\Sigma,\mathcal{X}}$.

▶ **Proposition 3.5.** $\mathsf{reg\text{-}}\mathfrak{S} = [\![\mathsf{reg\text{-}swm\text{-}}\mathfrak{L}]\!]$.

It is a straightforward, but important observation that for any given NFA over $\Sigma \cup \Gamma_{\mathcal{X}}$, we can efficiently check whether $\mathcal{L}(M)$ is a subword-marked language. Since most description mechanisms for regular languages (e. g., expressions, grammars, logics, etc.) easily translate into NFA, they can potentially all be used for defining regular spanners.

▶ **Proposition 3.6.** *Given an* NFA M *over alphabet* $\Sigma \cup \Gamma_{\mathcal{X}}$*, we can decide in time* $\mathrm{O}(|M||\mathcal{X}|^2)$ *if* $\mathcal{L}(M)$ *is a subword-marked language, and, if so, whether* $\mathcal{L}(M)$ *is functional in time* $\mathrm{O}(|M||\mathcal{X}|^2)$*, and whether it is hierarchical or quasi-disjoint in time* $\mathrm{O}(|M||\mathcal{X}|^3)$*.*

4 Refl-Spanners: Spanners with Built-In String-Equality Selections

In this section, we extend the concept of subword-marked words and languages in order to describe spanners with string-equality selections.

4.1 Ref-Words and Ref-Languages

We consider subword-marked words with extended terminal alphabet $\Sigma \cup \mathcal{X}$, i. e., in addition to symbols from Σ, also variables from $\mathcal{X}$ can appear as terminal symbols (the marking of subwords with symbols $\Gamma_{\mathcal{X}}$ remains unchanged).

▶ **Definition 4.1** (Ref-Words). *A* ref-word over Σ and $\mathcal{X}$ *is a subword-marked word over terminal alphabet* $\Sigma \cup \mathcal{X}$ *and variables* $\mathcal{X}$*, such that, for every* $\mathtt{x} \in \mathcal{X}$*, if* $w = w_1 \mathtt{x} w_2$*, then there exist words* v_1, v_2, v_3 *such that* $w_1 = v_1 \,{}^{\mathtt{x}}\!\triangleright\, v_2 \triangleleft^{\mathtt{x}} v_3$.

Since ref-words are subword-marked words, the properties "functional", "hierarchical" and "quasi-disjoint" are well-defined.

▶ **Example 4.2.** Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $\mathcal{X} = \{\mathtt{x}, \mathtt{y}\}$. The subword-marked word $u = \mathtt{ab}\,{}^{\mathtt{x}}\!\triangleright\,\mathtt{ab}\triangleleft^{\mathtt{x}}\,\mathtt{c}\,{}^{\mathtt{y}}\!\triangleright\,\mathtt{xaa}\triangleleft^{\mathtt{y}}\,\mathtt{y}$ and $v = \mathtt{a}\,{}^{\mathtt{x}}\!\triangleright\,\mathtt{ab}\,{}^{\mathtt{y}}\!\triangleright\,\mathtt{ab}\triangleleft^{\mathtt{x}}\,\mathtt{a}\triangleleft^{\mathtt{y}}\,\mathtt{xy}$ (over terminal alphabet $\Sigma \cup \mathcal{X}$ and variables $\mathcal{X}$) are valid ref-words (over terminal alphabet Σ and variables $\mathcal{X}$). Note that both u and v are functional, and u is also hierarchical, while v is not. On the other hand, $\mathtt{axb}\,{}^{\mathtt{x}}\!\triangleright\,\mathtt{ab}\triangleleft^{\mathtt{x}}\,\mathtt{c}\,{}^{\mathtt{y}}\!\triangleright\,\mathtt{xaa}\triangleleft^{\mathtt{y}}\,\mathtt{y}$ and $\mathtt{aa}\,{}^{\mathtt{x}}\!\triangleright\,\mathtt{ab}\triangleleft^{\mathtt{x}}\,\mathtt{c}\,{}^{\mathtt{y}}\!\triangleright\,\mathtt{ya}\triangleleft^{\mathtt{y}}$ are subword-marked words (over terminal alphabet $\Sigma \cup \mathcal{X}$ and variables $\mathcal{X}$), but not ref-words.

The idea of ref-words is that occurrences of $\mathtt{x} \in \mathcal{X}$ are interpreted as *references* to the subword ${}^{\mathtt{x}}\!\triangleright\, v \triangleleft^{\mathtt{x}}$, which we will call *definition* of $\mathtt{x}$. Note that while a single variable can have several references, there is at most one definition per variable, and if there is no definition for a variable, then it also has no references. Variable definitions may contain other references or definitions of other variables, i. e., there may be chains of references, e. g., the definition of $\mathtt{x}$ contains references of $\mathtt{y}$, and the definition of $\mathtt{y}$ contains references of $\mathtt{z}$ and so on. Next, we formally define this nested referencing process encoded by ref-words. Recall that for a subword-marked word w by $\mathfrak{e}(w)$ we denote the word obtained by removing all meta-symbols from $\Gamma_{\mathcal{X}}$ from w (however, if w is a ref-word, then $\mathfrak{e}(w)$ is a word over $\Sigma \cup \mathcal{X}$).

▶ **Definition 4.3** (Deref-Function). *For a ref-word* w *over* Σ *and* $\mathcal{X}$*, the subword-marked word* $\mathfrak{d}(w)$ *over* Σ *and* $\mathcal{X}$ *is obtained from* w *by repeating the following steps until we have a subword-marked word over* Σ *and* $\mathcal{X}$*:*

1. *Let* ${}^{\mathtt{x}}\!\triangleright\, v_{\mathtt{x}} \triangleleft^{\mathtt{x}}$ *be a definition such that* $\mathfrak{e}(v_{\mathtt{x}}) \in \Sigma^*$.
2. *Replace all occurrences of* $\mathtt{x}$ *in* w *by* $\mathfrak{e}(v_{\mathtt{x}})$.

It is straightforward to verify that the function $\mathfrak{d}(\cdot)$ is well-defined. By using this function and because ref-words encode subword-marked words, they can be interpreted as span-tuples. More precisely, for every ref-word w over Σ and $\mathcal{X}$, $\mathfrak{d}(w)$ is a subword-marked word over Σ and $\mathcal{X}$, $\mathfrak{e}(\mathfrak{d}(w)) \in \Sigma^*$ and $\mathsf{st}(\mathfrak{d}(w))$ is an $(\mathcal{X}, \mathfrak{e}(\mathfrak{d}(w)))$-tuple.

▶ **Example 4.4.** Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, let $\mathcal{X} = \{\mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3, \mathtt{x}_4\}$ and let

$$w = \mathtt{aa}\,{}^{\mathtt{x}_1}\!\triangleright\,\mathtt{ab}\,{}^{\mathtt{x}_2}\!\triangleright\,\mathtt{acc}\triangleleft^{\mathtt{x}_2}\,\mathtt{ax}_2\triangleleft^{\mathtt{x}_1}\,{}^{\mathtt{x}_4}\!\triangleright\,\mathtt{x}_1\mathtt{ax}_2\triangleleft^{\mathtt{x}_4}\,\mathtt{x}_4\mathtt{bx}_1\,.$$

Due to the definition ${}^{\mathtt{x}_2}\!\triangleright\,\mathtt{acc}\triangleleft^{\mathtt{x}_2}$, the procedure of Definition 4.3 will initially replace all references of $\mathtt{x}_2$ by $\mathtt{acc}$. Then, the definition for variable $\mathtt{x}_1$ is ${}^{\mathtt{x}_1}\!\triangleright\,\mathtt{ab}\,{}^{\mathtt{x}_2}\!\triangleright\,\mathtt{acc}\triangleleft^{\mathtt{x}_2}\,\mathtt{aacc}\triangleleft^{\mathtt{x}_1}$, so $\mathtt{abaccaacc}$ can be substituted for the references of $\mathtt{x}_1$. After replacing the last variable $\mathtt{x}_4$, we obtain

$$\mathfrak{d}(w) = \mathtt{aa}\,{}^{\mathtt{x}_1}\!\triangleright\,\mathtt{ab}\,{}^{\mathtt{x}_2}\!\triangleright\,\mathtt{acc}\triangleleft^{\mathtt{x}_2}\,\mathtt{aacc}\triangleleft^{\mathtt{x}_1}\,{}^{\mathtt{x}_4}\!\triangleright\,\mathtt{abaccaaccaacc}\triangleleft^{\mathtt{x}_4}\,\mathtt{abaccaaccaaccbabaccaacc}\,.$$

Moreover, we have $\mathsf{st}(\mathfrak{d}(w)) = ([3, 12\rangle, [5, 8\rangle, \bot, [12, 25\rangle)$.

As a non-hierarchical example, consider $u = {}^{x_1}\triangleright \mathsf{a}\, {}^{x_2}\triangleright \mathsf{aa} \triangleleft^{x_1} \mathsf{c}x_1\, {}^{x_3}\triangleright \mathsf{ac} \triangleleft^{x_2} x_2\mathsf{a}x_1 \triangleleft^{x_3}$. It can be easily verified that $\mathfrak{d}(u) = {}^{x_1}\triangleright \mathsf{a}\, {}^{x_2}\triangleright \mathsf{aa} \triangleleft^{x_1} \mathsf{caaa}\, {}^{x_3}\triangleright \mathsf{ac} \triangleleft^{x_2} \mathsf{aacaaaacaaaa} \triangleleft^{x_3}$ and $\mathsf{st}(\mathfrak{d}(u)) = ([1,4\rangle, [2,10\rangle, [8,22\rangle, \bot)$.

A set L of ref-words is called *ref-language* and we extend the $\mathfrak{d}(\cdot)$-function to ref-languages L in the obvious way, i. e., $\mathfrak{d}(L) = \{\mathfrak{d}(w) \mid w \in L\}$. As for subword-marked languages, we are especially interested in ref-languages that are regular. By $\mathsf{reg\text{-}ref\text{-}}\mathfrak{L}_{\Sigma,\mathcal{X}}$ we denote the class of regular ref-languages over Σ and $\mathcal{X}$, and we set $\mathsf{reg\text{-}ref\text{-}}\mathfrak{L} = \bigcup_{\Sigma,\mathcal{X}} \mathsf{reg\text{-}ref\text{-}}\mathfrak{L}_{\Sigma,\mathcal{X}}$.

In analogy to Proposition 3.6, we can easily check for a given NFA over alphabet $\Sigma \cup \Gamma_{\mathcal{X}} \cup \mathcal{X}$ whether it accepts a ref-language over Σ and $\mathcal{X}$.

▶ **Proposition 4.5.** *Given an* NFA M *where* $\mathcal{L}(M)$ *is a subword-marked language over* $\Sigma \cup \mathcal{X}$ *and* $\mathcal{X}$*, we can decide in time* $\mathrm{O}(|M||\mathcal{X}|^2)$ *if* $\mathcal{L}(M)$ *is a ref-language over* Σ *and* $\mathcal{X}$*.*

4.2 Refl-Spanners

We shall now define spanners based on regular ref-languages.

▶ **Definition 4.6** (Refl-Spanners)**.** *Let* L *be a ref-language (over* Σ *and* $\mathcal{X}$*). Then the* refl-spanner $[\![L]\!]_{\mathfrak{d}}$ *(over* Σ *and* $\mathcal{X}$*) is defined by* $[\![L]\!]_{\mathfrak{d}} = [\![\mathfrak{d}(L)]\!]$*. For a class* $\mathfrak{L}$ *of ref-languages, we set* $[\![\mathfrak{L}]\!]_{\mathfrak{d}} = \{[\![L]\!]_{\mathfrak{d}} \mid L \in \mathfrak{L}\}$*, and the class of* refl-spanners *is* $\mathsf{refl}\text{-}\mathfrak{S} = [\![\mathsf{reg\text{-}ref\text{-}}\mathfrak{L}]\!]_{\mathfrak{d}}$.

Since any regular ref-language L over Σ and $\mathcal{X}$ is also a regular subword-marked language over $\Sigma \cup \mathcal{X}$ and $\mathcal{X}$, $[\![L]\!]$ is, according to Definition 3.3, also a well-defined spanner (but over $\Sigma \cup \mathcal{X}$ and $\mathcal{X}$). However, whenever we are concerned with a ref-language L over Σ and $\mathcal{X}$ that is not also a subword-marked language over Σ and $\mathcal{X}$ (i. e., L contains actual occurrences of symbols from $\mathcal{X}$), then we are never interested in $[\![L]\!]$, but always in $[\![L]\!]_{\mathfrak{d}}$. Consequently, by a slight abuse of notation, we denote in this case $[\![L]\!]_{\mathfrak{d}}$ simply by $[\![L]\!]$.

For a regular ref-language L the corresponding refl-spanner $[\![L]\!]$ produces for a given $w \in \Sigma^*$ all $(\mathcal{X}, w)$-tuples t that are represented by some $u \in \mathfrak{d}(L)$ with $\mathfrak{e}(u) = w$, or, equivalently, all $(\mathcal{X}, w)$-tuples t with $t = \mathsf{st}(\mathfrak{d}(v))$ and $\mathfrak{e}(\mathfrak{d}(v)) = w$ for some $v \in L$. It is intuitively clear that the use of variable references of refl-spanners provide a functionality that resembles string-equality selections for core spanners. However, there are also obvious differences between these two spanner formalisms (as already mentioned in the introduction and as investigated in full detail in Section 6).

Before moving on to the actual results about refl-spanners, we shall briefly discuss another example.

▶ **Example 4.7.** Assume that we have a document $w = p_1 \# p_2 \# \ldots \# p_n$, where each $p_i \in \Sigma^*$ is the title page of a scientific paper and $\# \notin \Sigma$ is some separator symbol (e. g., a list of all title pages of papers in the issues of *Journal of the ACM* from 2000 to 2010). Let $\Sigma' = \Sigma \cup \{\#\}$. We define a refl-spanner

$$\alpha = \Sigma'^* \,\#\, \Sigma^*\, \mathsf{email}\colon\ {}^{x}\triangleright\, \Sigma^+ \triangleleft^{x}\, @\, r_{\mathsf{dom}}\, \Sigma^* \,\#\, \Sigma'^*\, \mathsf{email}\colon \mathsf{x}\, @\, r_{\mathsf{dom}}\, \Sigma^* \,\#\, \Sigma'^*\,,$$

where $r_{\mathsf{dom}} = \mathsf{hu\text{-}berlin.de} \vee \mathsf{tu\text{-}berlin.de} \vee \mathsf{fu\text{-}berlin.de}$ is a regular expressions that matches the email-domains of the three universities in Berlin. It can be easily seen that $[\![\alpha]\!](w)$ contains the first parts of the email-addresses of authors that have at least two JACM-papers between year 2000 and 2010 while working at a university in Berlin.

5 Evaluation and Static Analysis of Refl-Spanners

The problem ModelChecking is to decide whether $t \in S(w)$ for given spanner S over Σ and $\mathcal{X}$, $w \in \Sigma^*$ and $(\mathcal{X}, w)$-tuple t, and NonEmptiness is to decide $S(w) \neq \emptyset$ for given S and w. For the problems Satisfiability, Hierarchicality and Functionality, we get a single spanner S as input and ask whether there is a $w \in \Sigma^*$ with $S(w) \neq \emptyset$, whether S is hierarchical, or whether S is functional, respectively. Finally, Containment and Equivalence is to decide whether $S_1 \subseteq S_2$ or $S_1 = S_2$, respectively, for given spanners S_1 and S_2. The input refl-spanners are always given as NFA. Recall that a summary of our results is provided by Table 1 in the introduction.

▶ **Theorem 5.1.** ModelChecking *for* refl-$\mathfrak{S}$ *can be solved in time poly*$(|M|)(|w| + (2|\mathcal{X}|)!)$*, where* M *is an* NFA *that represents a refl-spanner* $S = [\![\mathcal{L}(M)]\!]$ *over* Σ *and* $\mathcal{X}$*,* $w \in \Sigma^*$*, and* t *is an* $(\mathcal{X}, w)$*-tuple. In case that* S *is functional or* M *is normalised (in the sense of Definition 5.4 that follows further below), this can be improved to poly*$(|M|)(|w| + |\mathcal{X}|)$ *and* $\mathrm{O}(|M|(|w| + |\mathcal{X}|))$*, respectively.*

Proof Sketch. To check $t \in [\![\mathcal{L}(M)]\!](w)$ it is sufficient to check whether there is some $v \in \mathcal{L}(M)$ with $\mathsf{st}(\mathfrak{d}(v)) = t$ and $\mathfrak{e}(\mathfrak{d}(v)) = w$. To this end, we turn w into a subword-marked word $\tilde{w}$ by inserting the symbols $\Gamma_{\mathcal{X}}$ according to t, but, since we do not know in which order the factors over $\Gamma_{\mathcal{X}}$ are read by M, we represent the maximal factors over $\Gamma_{\mathcal{X}}$ as sets (represented as single symbols) rather than words over $\Gamma_{\mathcal{X}}$. Moreover, for every $\mathsf{x} \in \mathcal{X}$, let u_{x} be the factor of w that corresponds to $t(\mathsf{x})$, if defined. Then we check whether M can accept $\tilde{w}$, but we treat x-transitions with $\mathsf{x} \in \mathcal{X}$ of M as labelled with u_{x}, and whenever we encounter a symbol that represents a subset of $\Gamma_{\mathcal{X}}$, then we have to compute in a brute-force manner which states are reachable by a path that reads exactly the symbols from this set (which causes the factor $(2|\mathcal{X}|)!$ in the running-time). Moreover, in order to not introduce another $|w|$ factor, we use a longest common extension data-structure to be able to consume prefixes u_{x} from $\tilde{w}$, i. e., to handle the x-transitions, in constant time.

If S is functional, then each two states q, q' of M uniquely determine which $u \in (\Gamma_{\mathcal{X}})^*$, if any, can be read while moving from q to q' (this observation has been used in a similar way by Freydenberger, Kimelfeld, and Peterfreund [9] for vset-automata). If M is normalised, we know exactly in which order the symbols from $\Gamma_{\mathcal{X}}$ have to be inserted into w in order to construct $\tilde{w}$. In both cases, this means that the brute-force part is not necessary. ◀

We discuss a few particularities about Theorem 5.1. The mentioned general running-time is not polynomial (in combined complexity), but nevertheless fixed-parameter tractable with respect to parameter $|\mathcal{X}|$. This is worth mentioning since for core spanners ModelChecking is W[1]-hard with respect to parameter $|\mathcal{X}|$ (this can be concluded from [8] and [5]). If M is already normalised (in the sense of Definition 5.4), then, in the likely case that $|\mathcal{X}| \in \mathrm{O}(|w|)$, we obtain a running-time of $\mathrm{O}(|M||w|)$ which is also known to be a lower-bound (subject to the Strong Exponential Time Hypothesis SETH) for regular expression matching (see [2] for further details), which obviously reduces to ModelChecking for refl-$\mathfrak{S}$.

We further obtain the following two theorems, which are proved by standard methods.

▶ **Theorem 5.2.** NonEmptiness *for* refl-$\mathfrak{S}$ *is* NP*-complete.*

▶ **Theorem 5.3.**
(a) Satisfiability *for* refl-$\mathfrak{S}$ *can be solved in time* $\mathrm{O}(|M|)$*,*
(b) Hierarchicality *for* refl-$\mathfrak{S}$ *can be solved in time* $\mathrm{O}(|M||\mathcal{X}|^3)$*, and*
(c) Functionality *for* refl-$\mathfrak{S}$ *can be solved in time* $\mathrm{O}(|M||\mathcal{X}|^2)$*,*
where M *is an* NFA *that describes a refl-spanner over* Σ *and* $\mathcal{X}$*.*

For core spanners, Satisfiability and Hierarchicality are PSpace-complete, even for restricted classes of core spanners (see [8]), and Containment and Equivalence are not semi-decidable (see [8]). We now show that for refl-spanners we can achieve decidability of Containment and Equivalence by imposing suitable restrictions. The goal is to reduce the containment of refl-spanners to the containment of their corresponding ref-languages, but the problem is that the correspondence between ref-words and the span-tuples they describe is not unique in 2 ways. (1): Different subword-marked words w and w' with $\mathfrak{e}(w) = \mathfrak{e}(w')$ can nevertheless describe the same span-tuple, i. e., $\mathsf{st}(w) = \mathsf{st}(w')$, since the order of consecutive occurrences of symbols from $\Gamma_{\mathcal{X}}$ has no impact on the represented span-tuple. And (2): The same subword-marked word can be the $\mathfrak{d}(\cdot)$-image of two different ref-words v and v', i. e., $\mathfrak{d}(v) = \mathfrak{d}(v')$, e. g., the ref-words $w_1 = {}^{x}\triangleright\mathtt{ab}\triangleleft^{x}\mathtt{b}\,{}^{y}\triangleright\mathtt{abb}\triangleleft^{y}\mathtt{yab}$, $w_2 = {}^{x}\triangleright\mathtt{ab}\triangleleft^{x}\mathtt{b}\,{}^{y}\triangleright\mathtt{xb}\triangleleft^{y}\mathtt{xbx}$ and $w_3 = {}^{x}\triangleright\mathtt{ab}\triangleleft^{x}\mathtt{b}\,{}^{y}\triangleright\mathtt{abb}\triangleleft^{y}\mathtt{yx}$ are all $\preceq$-normalised (in the sense of Definition 5.4 that follows further below), where ${}^{x}\triangleright \preceq \triangleleft^{x} \preceq {}^{y}\triangleright \preceq \triangleleft^{y}$. However, $\mathfrak{d}(w_1) = \mathfrak{d}(w_2) = \mathfrak{d}(w_3) = {}^{x}\triangleright\,\mathtt{ab}\,\triangleleft^{x}\,\mathtt{b}\,{}^{y}\triangleright\,\mathtt{abb}\,\triangleleft^{y}\,\mathtt{abbab}$.

We can deal with issue (1) by requiring for any two subword-marked languages L and L' that all consecutive occurrences of symbols from $\Gamma_{\mathcal{X}}$ are ordered in the same way, since then $[\![L]\!] \subseteq [\![L']\!]$ is characterised by $L \subseteq L'$. This is actually a rephrasing of an analogous result about vset-automata from [3]. However, issue (2) is independent of the order of symbols from $\Gamma_{\mathcal{X}}$, i. e., even if we assume that the symbols from $\Gamma_{\mathcal{X}}$ are ordered in the same way in two ref-languages L and L', it is still not necessarily the case that $[\![L]\!] \subseteq [\![L']\!]$ is characterised by $L \subseteq L'$. In order to deal with issue (2) we need to impose some actual restrictions on ref-languages. Intuitively speaking, we require all variable references to be extracted by their own private extraction variable, i. e., in the ref-words we encounter all variable references x in the form ${}^{y_{\mathsf{x}}}\triangleright\,\mathsf{x}\triangleleft^{y_{\mathsf{x}}}$, where y_{x} has in all ref-words the sole purpose of extracting the content of some reference of variable x. With this requirement, the positions of the repeating factors described by variables and their references must be explicitly present as spans in the span-tuples. This seems like a strong restriction for refl-spanners, but we should note that for core spanners we necessarily have a rather similar situation: if we want to use string equality selections on some spans, we have to explicitly extract them by variables first.

Before we formally define the restriction of ref-languages mentioned above, we first develop some technical tools on the level of subword-marked languages.

▶ **Definition 5.4.** *A linear order $\preceq$ on $\Gamma_{\mathcal{X}}$ is* valid *if, for every $\mathsf{x} \in \mathcal{X}$, ${}^{x}\triangleright \preceq \triangleleft^{x}$. A subword-marked word w over Σ and $\mathcal{X}$ is* $\preceq$-normalised *for a valid linear order $\preceq$ on $\Gamma_{\mathcal{X}}$, if, for every $\sigma_1, \sigma_2 \in \Gamma_{\mathcal{X}}$ and every factor $\sigma_1\sigma_2$ in w, we have that $\sigma_1 \preceq \sigma_2$. A subword-marked word over Σ and $\mathcal{X}$ is* normalised*, if it is $\preceq$-normalised for some valid linear order $\preceq$ on $\Gamma_{\mathcal{X}}$.*

▶ **Example 5.5.** Let $\mathcal{X} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$ and let $\preceq$ be the valid order on $\Gamma_{\mathcal{X}}$ with ${}^{x}\triangleright \preceq {}^{y}\triangleright \preceq {}^{z}\triangleright \preceq \triangleleft^{x} \preceq \triangleleft^{y} \preceq \triangleleft^{z}$. Then $\mathtt{a}\,{}^{x}\triangleright\,{}^{y}\triangleright\,\mathtt{ab}\,\triangleleft^{y}\,{}^{z}\triangleright\,\triangleleft^{x}\,\mathtt{b}\triangleleft^{z}$ and ${}^{z}\triangleright\,{}^{x}\triangleright\,\mathtt{ab}\,\triangleleft^{z}\,\mathtt{a}\triangleleft^{x}$ are *not* $\preceq$-normalised. On the other hand, $\mathtt{a}\,{}^{x}\triangleright\,{}^{y}\triangleright\,\mathtt{ab}\,{}^{z}\triangleright\,\triangleleft^{x}\,\triangleleft^{y}\,\mathtt{b}\triangleleft^{z}$ and ${}^{z}\triangleright\,\mathtt{a}\,{}^{x}\triangleright\,\mathtt{ab}\,\triangleleft^{z}\,\mathtt{a}\triangleleft^{x}$ are $\preceq$-normalised.

▶ **Lemma 5.6.** *Let w_1 and w_2 be subword-marked words over Σ and $\mathcal{X}$ that are $\preceq$-normalised for a valid linear order $\preceq$ and that satisfy $\mathfrak{e}(w_1) = \mathfrak{e}(w_2)$. Then $\mathsf{st}(w_1) = \mathsf{st}(w_2)$ if and only if $w_1 = w_2$.*

This concept of normalised subword-marked words directly carries over to subword-marked languages: a subword-marked language L over Σ and $\mathcal{X}$ is *$\preceq$-normalised*, if every $w \in L$ is $\preceq$-normalised; and a subword-marked language over Σ and $\mathcal{X}$ is *normalised*, if it is $\preceq$-normalised for some valid linear order $\preceq$ on $\Gamma_{\mathcal{X}}$. Note that we do not only require all words of L to be normalised, but to be normalised with respect to the same valid order $\preceq$. And since ref-words are subword-marked words (and ref-languages are subword-marked languages), the concept of $\preceq$-normalisation also applies to ref-words and ref-languages.

Obviously, if L and K are $\preceq$-normalised subword-marked languages over Σ and $\mathcal{X}$, then $L \cup K$ is a $\preceq$-normalised subword-marked languages over Σ and $\mathcal{X}$.

For a given subword-marked language L over Σ and $\mathcal{X}$, and a valid order $\preceq$, we can obtain a $\preceq$-normalised subword-marked language L' over Σ and $\mathcal{X}$ with $[\![L]\!] = [\![L']\!]$ by simply reordering all maximal factors over $\Gamma_{\mathcal{X}}$ in the words of L according to $\preceq$. The next lemma shows how to effectively do this if L is regular and represented by an NFA M.

▶ **Lemma 5.7.** *Let M be an* NFA *that accepts a subword-marked language over Σ and $\mathcal{X}$, and let $\preceq$ be a valid linear order on $\Gamma_{\mathcal{X}}$. Then we can compute in time $\mathrm{O}(2^{|\Gamma_{\mathcal{X}}|}|M|)$ an* NFA *M' of size $\mathrm{O}(2^{|\Gamma_{\mathcal{X}}|}|M|)$ such that $[\![\mathcal{L}(M)]\!] = [\![\mathcal{L}(M')]\!]$ and $\mathcal{L}(M')$ is a $\preceq$-normalised subword-marked language over Σ and $\mathcal{X}$.*

The general idea for the lemma's proof is that whenever M reads a sequence of symbols from $\Gamma_{\mathcal{X}}$, then M' instead stores the letters of this sequence in the finite state control and makes ε-transitions instead. Whenever M stops reading the $\Gamma_{\mathcal{X}}$-sequence and makes the next Σ-transition, then M' first reads the recorded symbols from $\Gamma_{\mathcal{X}}$ from the input, but according to the order $\preceq$, and then reads the next Σ-transition.

From Lemmas 5.6 and 5.7, we can directly derive a procedure for deciding Containment for regular spanners: given $S_1, S_2 \in \mathsf{reg}\text{-}\mathfrak{S}_{\Sigma,\mathcal{X}}$ represented by NFA M_1 and M_2, choose an arbitrary valid order $\preceq$ on $\mathcal{X}$ and compute NFA M_1' and M_2' according to Lemma 5.7. Now Lemma 5.6 directly implies that $[\![\mathcal{L}(M_1)]\!] \subseteq [\![\mathcal{L}(M_2)]\!]$ if and only if $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$. An analogue of Lemmas 5.6 and 5.7 on the level of vset-automata is given in [3], although without stating an explicit upper complexity bound for the construction (moreover, a similar construction (without complexity bounds) is also used in [4] to show that regular spanners are closed under the join operator).

We next turn to the problem that different ref-words w_1 and w_2, even though both $\preceq$-normalised, can describe the same span-tuple, simply because $\mathfrak{d}(w_1) = \mathfrak{d}(w_2)$. For example, the ref-words $w_1 = {}^{\mathsf{x}}\triangleright\,\mathtt{ab}\,\triangleleft^{\mathsf{x}}\,\mathtt{b}\,{}^{\mathsf{y}}\triangleright\,\mathtt{abb}\,\triangleleft^{\mathsf{y}}\,\mathtt{yab}$, $w_2 = {}^{\mathsf{x}}\triangleright\,\mathtt{ab}\,\triangleleft^{\mathsf{x}}\,\mathtt{b}\,{}^{\mathsf{y}}\triangleright\,\mathtt{xb}\,\triangleleft^{\mathsf{y}}\,\mathtt{xbx}$ and $w_3 = {}^{\mathsf{x}}\triangleright\,\mathtt{ab}\,\triangleleft^{\mathsf{x}}\,\mathtt{b}\,{}^{\mathsf{y}}\triangleright\,\mathtt{abb}\,\triangleleft^{\mathsf{y}}\,\mathtt{yx}$ are all $\preceq$-normalised, where ${}^{\mathsf{x}}\triangleright \preceq \triangleleft^{\mathsf{x}} \preceq {}^{\mathsf{y}}\triangleright \preceq \triangleleft^{\mathsf{y}}$. However, $\mathfrak{d}(w_1) = \mathfrak{d}(w_2) = \mathfrak{d}(w_3) = {}^{\mathsf{x}}\triangleright\,\mathtt{ab}\,\triangleleft^{\mathsf{x}}\,\mathtt{b}\,{}^{\mathsf{y}}\triangleright\,\mathtt{abb}\,\triangleleft^{\mathsf{y}}\,\mathtt{abbab}$.

It is our goal to restrict ref-words in such a way that $w_1 \neq w_2$ implies $\mathfrak{d}(w_1) \neq \mathfrak{d}(w_2)$. We first explain this restriction on an intuitive level. The set of variables $\mathcal{X}$ is partitioned into a set $\mathcal{X}_r$ of *reference-variables* and, for each such reference-variable $\mathsf{x} \in \mathcal{X}_r$, into a set $\mathcal{X}_{e,\mathsf{x}}$ of *extraction-variables*. For every $\mathsf{x} \in \mathcal{X}$, every reference of x is extracted by some $\mathsf{y} \in \mathcal{X}_{e,\mathsf{x}}$, i. e., it occurs between ${}^{\mathsf{y}}\triangleright$ and $\triangleleft^{\mathsf{y}}$; moreover, every $\mathsf{y} \in \mathcal{X}_{e,\mathsf{x}}$ either does not occur at all, or it is used as extractor for x, i. e., y's definition contains exactly one occurrence of a symbol $\Sigma \cup \mathcal{X}$ which is x. In addition, we also require that for every $\mathsf{x} \in \mathcal{X}_r$ with at least one reference, the image of x under $\mathfrak{d}(\cdot)$ is non-empty (otherwise the deref-function may turn normalised ref-words into non-normalised ones by joining two previously separate factors over $\Gamma_{\mathcal{X}}$).

▶ **Definition 5.8.** *A ref-word w over Σ and $\mathcal{X}$ is a* strongly reference extracting *ref-word over Σ and $(\mathcal{X}_r, \{\mathcal{X}_{e,\mathsf{x}} \mid \mathsf{x} \in \mathcal{X}_r\})$, if it satisfies the following:*

- *$\mathcal{X} = \mathcal{X}_r \cup \bigcup_{\mathsf{x}\in\mathcal{X}_r} \mathcal{X}_{e,\mathsf{x}}$, where all sets $\mathcal{X}_r$ and $\mathcal{X}_{e,\mathsf{x}}$ with $\mathsf{x} \in \mathcal{X}_r$ are pairwise disjoint.*
- *Each reference of an $\mathsf{x} \in \mathcal{X}_r$ in w occurs in a factor ${}^{\mathsf{y}}\triangleright\,\gamma\mathsf{x}\delta\triangleleft^{\mathsf{y}}$ with $\mathsf{y} \in \mathcal{X}_{e,\mathsf{x}}$, $\gamma, \delta \in (\Gamma_{\mathcal{X}})^*$.*
- *If an $\mathsf{y} \in \mathcal{X}_{e,\mathsf{x}}$ has a definition in w, then it has the form ${}^{\mathsf{y}}\triangleright\,\gamma\mathsf{x}\delta\triangleleft^{\mathsf{y}}$ with $\gamma, \delta \in (\Gamma_{\mathcal{X}})^*$, and $|w|_{\mathsf{y}} = 0$.*
- *For every $\mathsf{x} \in \mathcal{X}_r$ with $|w|_{\mathsf{x}} \neq 0$, $\mathsf{st}(\mathfrak{d}(w))(\mathsf{x}) = [i, j\rangle$ with $j - i \geqslant 2$.*

A ref-language L over Σ and $\mathcal{X}$ is a *strongly reference extracting* ref-language over Σ and $(\mathcal{X}_r, \{\mathcal{X}_{e,\mathsf{x}} \mid \mathsf{x} \in \mathcal{X}_r\})$ if every ref-word $w \in L$ is a strongly reference extracting ref-word over Σ and $(\mathcal{X}_r, \{\mathcal{X}_{e,\mathsf{x}} \mid \mathsf{x} \in \mathcal{X}_r\})$. By a series of intermediate results we obtain:

▶ **Lemma 5.9.** *Let L, K be two strongly reference extracting and $\preceq$-normalised ref-languages over Σ and $(\mathcal{X}_r, \{\mathcal{X}_{e,\mathsf{x}} \mid \mathsf{x} \in \mathcal{X}_r\})$. Then $[\![L]\!] \subseteq [\![K]\!]$ if and only if $L \subseteq K$.*

Let sre-refl-$\mathfrak{S}_{\Sigma,(\mathcal{X}_r,\{\mathcal{X}_{e,\mathsf{x}}|\mathsf{x}\in\mathcal{X}_r\})}$ be the class of strongly reference extracting refl-spanners over Σ and $(\mathcal{X}_r, \{\mathcal{X}_{e,\mathsf{x}} \mid \mathsf{x} \in \mathcal{X}_r\})$.

▶ **Theorem 5.10.** Containment *and* Equivalence *for* sre-refl-$\mathfrak{S}_{\Sigma,(\mathcal{X}_r,\{\mathcal{X}_{e,\mathsf{x}}|\mathsf{x}\in\mathcal{X}_r\})}$ *is in* ExpSpace *if the refl-spanners are given as* NFA*, in* PSpace *if the refl-spanners are given as $\preceq$-normalised* NFA*, and in* NLogSpace *if the refl-spanners S_1 and S_2 are given as $\preceq$-normalised* DFA*.*

Proof Sketch. Let M_1 and M_2 be the NFA that represent S_1 and S_2. For deciding Containment we first turn M_1 and M_2 into normalised variants M_1' and M_2' with Lemma 5.7 (which requires exponential space) and then check whether $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$ (which is known to be possible in polynomial space for NFA and nondeterministic logarithmic space for DFA). The correctness is a direct consequence of Lemma 5.9. This also shows the other bounds. ◀

6 Expressive Power of Refl-Spanners

It is a straightfoward observation that the expressive power of refl-spanners properly exceeds the one of regular spanners, but it is less clear which refl-spanners are also core spanners and which core spanners are refl-spanners. We first briefly discuss the former question.

A ref-language L over Σ and $\mathcal{X}$ is *reference-bounded* if there is a number k with $|w|_{\mathsf{x}} \leqslant k$ for every $\mathsf{x} \in \mathcal{X}$ and every $w \in L$. A refl-spanner is *reference-bounded* if it is represented by a reference-bounded ref-language. The following is an easy exercise.

▶ **Theorem 6.1.** *Every reference-bounded refl-spanner is a core spanner.*

It is interesting to note that the not reference-bounded refl-spanner $[\![\mathcal{L}(\mathsf{a}^+ \,{}^{\mathsf{x}}\!\triangleright\, \mathsf{b}^+ \triangleleft^{\mathsf{x}} (\mathsf{a}^+\mathsf{x})^*\mathsf{a}^+)]\!]$ is provably not a core spanner (see [4, Theorem 6.1])).

The question which core spanners can be represented as refl-spanners is a much more difficult one and we shall investigate it in more detail. There are simple core spanners which translate to refl-spanners in an obvious way, e. g., $\pi_{\{\mathsf{x}\}}\zeta^{=}_{\{\mathsf{x},\mathsf{y}\}}[\![L]\!]$ with $L = \mathcal{L}(\,{}^{\mathsf{x}}\!\triangleright (\mathsf{a}^* \vee \mathsf{b}^*) \triangleleft^{\mathsf{x}} \mathsf{c}\, {}^{\mathsf{y}}\!\triangleright (\mathsf{a}^* \vee \mathsf{b}^*) \triangleleft^{\mathsf{y}})$ can be represented as $[\![L']\!]$ where $L' = \mathcal{L}(\,{}^{\mathsf{x}}\!\triangleright (\mathsf{a}^* \vee \mathsf{b}^*) \triangleleft^{\mathsf{x}} \mathsf{cx})$. However, if we change L to $\mathcal{L}(\,{}^{\mathsf{x}}\!\triangleright \Sigma^*\mathsf{a}\Sigma^* \triangleleft^{\mathsf{x}} \mathsf{c}\, {}^{\mathsf{y}}\!\triangleright \Sigma^*\mathsf{b}\Sigma^*\triangleleft^{\mathsf{y}})$, then neither ref-language $\mathcal{L}(\,{}^{\mathsf{x}}\!\triangleright \Sigma^*\mathsf{a}\Sigma^* \triangleleft^{\mathsf{x}} \mathsf{cx})$ nor $\mathcal{L}(\,{}^{\mathsf{x}}\!\triangleright \Sigma^*\mathsf{b}\Sigma^* \triangleleft^{\mathsf{x}} \mathsf{cx})$ yield an equivalent refl-spanner and we have to use $\mathcal{L}(\,{}^{\mathsf{x}}\!\triangleright r \triangleleft^{\mathsf{x}} \mathsf{cx})$, where r is a regular expression for $\mathcal{L}(\Sigma^*\mathsf{a}\Sigma^*) \cap \mathcal{L}(\Sigma^*\mathsf{b}\Sigma^*)$.

Another problem is that core spanners can also use string-equality selections on spans that contain start or end positions of other spans. For example, it seems difficult to transform $\zeta^{=}_{\{\mathsf{x},\mathsf{y}\}}[\![\mathcal{L}(\,{}^{\mathsf{x}}\!\triangleright \mathsf{a}^* \triangleleft^{\mathsf{x}}\, {}^{\mathsf{y}}\!\triangleright\, {}^{\mathsf{z}}\!\triangleright \mathsf{a}^* \triangleleft^{\mathsf{z}} \mathsf{a}^* \triangleleft^{\mathsf{y}})]\!]$ into a refl-spanner. The situation gets even more involved if we use the string-equality selections directly on overlapping spans, e. g., as in core spanners of the form $\zeta^{=}_{\{\mathsf{x},\mathsf{y}\}}([\![\mathcal{L}(\,{}^{\mathsf{x}}\!\triangleright \ldots\, {}^{\mathsf{y}}\!\triangleright \ldots \triangleleft^{\mathsf{x}} \ldots \triangleleft^{\mathsf{y}})]\!])$. For an in-depth analysis of the capability of core spanners to describe word-combinatorial properties, we refer to [8, 7].

These considerations suggest that the refl-spanner formalism is much less powerful than core spanners, which is to be expected, since we have to pay a price for the fact that we can solve many problems for refl-spanners much more efficiently than for core spanners (see our results presented in Section 5 and summarised in Table 1). However, we can show that a surprisingly large class of core spanners can nevertheless be represented by a single refl-spanner along with the application of a simple spanner operation (to be defined next) that just combines several variables (or columns in the spanner result) into one variable (or column) in a natural way, and a projection.

The *span-fusion* $\uplus$ is a binary operation $\mathsf{Spans} \times \mathsf{Spans} \to \mathsf{Spans}$ defined by $[i,j\rangle \uplus [i',j'\rangle = [\min\{i,i'\}, \max\{j,j'\}\rangle$ and $[i,j\rangle \uplus \bot = \bot \uplus [i,j\rangle = [i,j\rangle$. For a set $K \subseteq \mathsf{Spans}(w)$, we define $\biguplus(K) = \bot$ if $K = \emptyset$ and $\biguplus(K) = \biguplus(K \setminus \{s\}) \uplus s$ if $s \in K$. Intuitively speaking, the operation $\uplus$ constructs the set-union of two spans and fills in the gaps to turn it into a valid span.

We next lift this operation to an operation on spanners with the following intended meaning. In a table $S(w)$ for some spanner S over Σ and $\mathcal{X}$, and $w \in \Sigma^*$, we want to replace a specified set of columns $\{\mathsf{y}_1, \mathsf{y}_2, \ldots, \mathsf{y}_k\} \subseteq \mathcal{X}$ by a single new column x that, for each row s (i. e., span-tuple s) in $S(w)$, contains the span $\biguplus(\{s(\mathsf{y}_i) \mid i \in [k]\})$.

▶ **Definition 6.2.** *Let $\lambda \subseteq \mathcal{X}$ and let x be a new variable with $\mathsf{x} \notin \mathcal{X} \setminus \lambda$. For any $\mathcal{X}$-tuple t, $\biguplus_{\lambda \to \mathsf{x}}(t)$ is the $((\mathcal{X} \setminus \lambda) \cup \{\mathsf{x}\})$-tuple with $(\biguplus_{\lambda \to \mathsf{x}}(t))(\mathsf{x}) = \biguplus(\{t(\mathsf{y}) \mid \mathsf{y} \in \lambda\})$ and $(\biguplus_{\lambda \to \mathsf{x}}(t))(\mathsf{z}) = t(\mathsf{z})$ for every $\mathsf{z} \in \mathcal{X} \setminus \lambda$. For a set R of $\mathcal{X}$-tuples, $\biguplus_{\lambda \to \mathsf{x}}(R) = \{\biguplus_{\lambda \to \mathsf{x}}(t) \mid t \in R\}$. Moreover, for a spanner S over Σ and $\mathcal{X}$, the spanner $\biguplus_{\lambda \to \mathsf{x}}(S)$ over Σ and $(\mathcal{X} \cup \{\mathsf{x}\})$ is defined by $(\biguplus_{\lambda \to \mathsf{x}}(S))(w) = \biguplus_{\lambda \to \mathsf{x}}(S(w))$ for every word w.*

We use the following generalised application of the operation $\biguplus_{\lambda \to \mathsf{x}}$. For $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\} \subseteq \mathcal{P}(\mathcal{X})$ such that all λ_i with $i \in [k]$ are pairwise disjoint, a spanner S over $\mathcal{X}$ and fresh variables $\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_k$, we define $\biguplus_{\{\lambda_i \to \mathsf{x}_i \mid i \in [k]\}}(S) = \biguplus_{\lambda_1 \to \mathsf{x}_1}(\biguplus_{\lambda_2 \to \mathsf{x}_2}(\ldots \biguplus_{\lambda_k \to \mathsf{x}_k}(S) \ldots))$. If the new variables x_i are negligible or clear from the context, we also write $\biguplus_{\lambda}$ or $\biguplus_{\{\lambda_i \mid i \in [k]\}}$ instead of $\biguplus_{\lambda \to \mathsf{x}}$ or $\biguplus_{\{\lambda_i \to \mathsf{x}_i \mid i \in [k]\}}$.

▶ **Example 6.3.** Let $L = \mathcal{L}(\, {}^{\mathsf{x}}\triangleright\, \mathsf{a}^*\, {}^{\mathsf{y}}\triangleright\, \mathsf{b}^*\, \triangleleft^{\mathsf{x}}\, \mathsf{a}^*\triangleleft^{\mathsf{y}})$ be a non-hierarchical subword-marked language over $\Sigma = \{\mathsf{a}, \mathsf{b}\}$ and $\mathcal{X} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$. For $w = \mathsf{aabaaa}$, we have $[\![L]\!](w) = \{([1,4\rangle, [3,7\rangle)\}$. Moreover, let $L' = \mathcal{L}(\, {}^{\mathsf{x}}\triangleright\, \mathsf{a}^*\, \triangleleft^{\mathsf{x}}\, {}^{\mathsf{y}}\triangleright\, \mathsf{b}^*\, \triangleleft^{\mathsf{y}}\, {}^{\mathsf{z}}\triangleright\, \mathsf{a}^*\triangleleft^{\mathsf{z}})$ be a *hierarchical* subword-marked language, and let $\lambda_1 = \{\mathsf{x}, \mathsf{y}\}$, $\lambda_2 = \{\mathsf{y}, \mathsf{z}\}$ and $\Lambda = \{\lambda_1, \lambda_2\}$. Then $\biguplus_{\Lambda}[\![L']\!](w) = \{\biguplus_{\Lambda}(([1,3\rangle, [3,4\rangle, [4,7\rangle))\} = [\![L]\!](w)$. In fact, it can be easily verified that $\biguplus_{\Lambda}[\![L']\!] = [\![L]\!]$.

Two variables $\mathsf{x}, \mathsf{y} \in \mathcal{X}$ are *overlapping in a spanner* S over Σ and $\mathcal{X}$ if there is a $w \in \Sigma^*$ and $t \in S(w)$ such that $t(\mathsf{x}) \cap t(\mathsf{y}) \neq \emptyset$. A string equality-selection $\varsigma^{=}_{E}$ over $\mathcal{X}$ is *overlapping with respect to* S if there are variables $\mathsf{x}, \mathsf{y} \in \bigcup_{\mathcal{Z} \in E} \mathcal{Z}$ with $\mathsf{x} \neq \mathsf{y}$ that are overlapping in S. A string-equality selection $\varsigma^{=}_{E}$ is *non-overlapping with respect to* S if it is not overlapping with respect to S. We are now able to state the main result of this section:

▶ **Theorem 6.4.** *For every core spanner S with $S = \pi_{\mathcal{Y}}\varsigma^{=}_{E}(S')$, where S' is a regular spanner over Σ and $\mathcal{X}$, $\mathcal{Y} \subseteq \mathcal{X}$ and $E \subseteq \mathcal{P}(\mathcal{X})$ such that $\varsigma^{=}_{E}$ is a string-equality selection over $\mathcal{X}$ that is non-overlapping with respect to S', there is a reference-bounded refl-spanner S'' over Σ and $\mathcal{X}'$ with $|\mathcal{X}'| = \mathrm{O}(|\mathcal{X}|^3)$ and a set $\Lambda \subseteq \mathcal{P}(\mathcal{X}')$ such that $S = \pi_{\mathcal{Y}} \biguplus_{\Lambda}(S'')$.*

We discuss this result before giving a proof sketch. The core simplification lemma states that in every core spanner $S \in \mathsf{reg}\text{-}\mathfrak{S}^{\{\cup, \pi, \bowtie, \varsigma^{=}\}}$, we can "push" all applications of $\cup$ and $\bowtie$ into the NFA that represents the regular spanner, leaving us with an expression $\pi_{\mathcal{Y}}\varsigma^{=}_{E}(\mathcal{L}(M))$ for an NFA M that accepts a subword-marked language. Theorem 6.4 now assures that if $\varsigma^{=}_{E}$ is non-overlapping with respect to S, then we can further "push" even all string-equality selections into the NFA M, which then accepts a (reference-bounded) ref-language instead of a subword-marked language, i. e., we can represent the string-equality selections as mere variable references in the regular spanner representation. However, the construction will add (a polynomial number of) new variables, which have to be translated back by an application of the span-fusion.

A main building block for the proof of Theorem 6.4, which also constitutes an interesting result about core spanners in its own right, is the following normal-form result.

▶ **Theorem 6.5.** *Every core spanner S over Σ and $\mathcal{X}$ can be represented as $\pi_{\mathcal{Y}}(\biguplus_{\Lambda}(\varsigma^{=}_{E}(S')))$, where S' is a quasi-disjoint regular spanner over Σ and $\mathcal{X}'$ with $|\mathcal{X}'| = \mathrm{O}(|\mathcal{X}|^3)$, $\mathcal{Y} \subseteq \mathcal{X}$, $E, \Lambda \subseteq \mathcal{P}(\mathcal{X}')$.*

Proof Sketch. Due to the core-simplification lemma (Lemma 2.2), we can assume that $S = \pi_{\mathcal{Y}}\varsigma^{=}_{E}(S')$ for some regular spanner S' over Σ and $\mathcal{X}$, and $E = \{\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_\ell\} \subseteq \mathcal{P}(\mathcal{X})$. Let M be an NFA with $[\![\mathcal{L}(M)]\!] = S'$. We first transform M into M' by replacing every $\mathsf{x} \in \mathcal{X}$ by $m = |\mathcal{X}|^2$ new variables $\mathcal{SV}_m(\mathsf{x}) = \{[\mathsf{x},1], [\mathsf{x},2], \ldots, [\mathsf{x},m]\}$, i. e., M' has variable set $\mathcal{X}' = \bigcup^{\mathsf{x}\in\mathcal{X}} \mathcal{SV}_m(\mathsf{x})$. Whenever M reads a factor ${}^{\mathsf{x}}\!\triangleright w_{\mathsf{x}} \triangleleft^{\mathsf{x}}$, then M' nondeterministically reads a factor ${}^{[\mathsf{x},1]}\!\triangleright w_{\mathsf{x},1} \triangleleft^{[\mathsf{x},1]}\ {}^{[\mathsf{x},2]}\!\triangleright w_{\mathsf{x},2} \triangleleft^{[\mathsf{x},2]} \ldots {}^{[\mathsf{x},m]}\!\triangleright w_{\mathsf{x},m} \triangleleft^{[\mathsf{x},m]}$ instead, where $w_{\mathsf{x},1}w_{\mathsf{x},2}\ldots w_{\mathsf{x},m}$ is a factorisation of w_{x}, such that, for every $j \in [m]$, $w_{\mathsf{x},j} \in (\Gamma_{\mathcal{X}'})^*\Sigma^*(\Gamma_{\mathcal{X}'})^*$. This can be done by keeping track in the finite state control for which $j \in [m]$ we have already read ${}^{[\mathsf{x},j]}\!\triangleright$ and then allowing M' to nondeterministically read factors $\triangleleft^{[\mathsf{x},j]}\ {}^{[\mathsf{x},j+1]}\!\triangleright$ while reading w_{x}. In order to make sure that $w_{\mathsf{x},j} \in (\Gamma_{\mathcal{X}'})^*\Sigma^*(\Gamma_{\mathcal{X}'})^*$ for every $j \in [m]$, we have to require that at least one such $\triangleleft^{[\mathsf{x},j]}\ {}^{[\mathsf{x},j+1]}\!\triangleright$ factor is read in every maximal factor of symbols from $\Gamma_{\mathcal{X}\setminus\{\mathsf{x}\}}$.

It can be shown that $\mathcal{L}(M')$ is in fact a quasi-disjoint subword-marked language over Σ and $\mathcal{X}'$. Moreover, we can express $\varsigma^{=}_{\mathcal{Z}_1}\varsigma^{=}_{\mathcal{Z}_2}\ldots\varsigma^{=}_{\mathcal{Z}_\ell}([\![\mathcal{L}(M)]\!])$ by turning each $\varsigma^{=}_{\mathcal{Z}_i}$ into string-equality selections $\varsigma^{=}_{E_i}$ with $E_i = \{\{[\mathsf{x},1] \mid \mathsf{x} \in \mathcal{Z}_i\}, \{[\mathsf{x},2] \mid \mathsf{x} \in \mathcal{Z}_i\}, \ldots, \{[\mathsf{x},m] \mid \mathsf{x} \in \mathcal{Z}_i\}\}$ and applying them to $[\![\mathcal{L}(M)]\!]$, followed by a span-fusion that will combine back all variables from $\mathcal{SV}_m(\mathsf{x})$ into the single original variable x. More precisely, $\varsigma^{=}_{\mathcal{Z}_1}\varsigma^{=}_{\mathcal{Z}_2}\ldots\varsigma^{=}_{\mathcal{Z}_\ell}([\![\mathcal{L}(M)]\!]) = \biguplus_{\Lambda}\varsigma^{=}_{E_1}\varsigma^{=}_{E_2}\ldots\varsigma^{=}_{E_\ell}([\![\mathcal{L}(M')]\!])$, where $\Lambda = \{\mathcal{SV}_m(\mathsf{x}) \to \mathsf{x} \mid \mathsf{x} \in \mathcal{X}\}$. ◀

We are now ready to give a proof sketch for Theorem 6.4:

Proof Sketch of Theorem 6.4. Let $S = \pi_{\mathcal{Y}}\varsigma^{=}_{E}(S')$, where S' is a regular spanner over Σ and $\mathcal{X}$, $\mathcal{Y} \subseteq \mathcal{X}$ and $E = \{\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_\ell\} \subseteq \mathcal{P}(\mathcal{X})$ such that $\varsigma^{=}_{E}$ is non-overlapping with respect to S'. Let M be an NFA with $[\![\mathcal{L}(M)]\!] = S'$. The proof heavily relies on Theorem 6.5 in the sense that we only have to show that $\varsigma^{=}_{E_1}\varsigma^{=}_{E_2}\ldots\varsigma^{=}_{E_\ell}([\![\mathcal{L}(M')]\!])$ is a refl-spanner, where M' is obtained from M via the construction of the proof of Theorem 6.5. In particular, $\mathcal{L}(M')$ is quasi-disjoint and the string-equality selections $\varsigma^{=}_{E_1}, \varsigma^{=}_{E_2}, \ldots, \varsigma^{=}_{E_\ell}$ have been obtained from a string-equality selection $\varsigma^{=}_{E}$ that is non-overlapping with respect to S'.

We recall that, for every $i \in [\ell]$, $E_i = \{\{[\mathsf{x},1] \mid \mathsf{x} \in \mathcal{Z}_i\}, \{[\mathsf{x},2] \mid \mathsf{x} \in \mathcal{Z}_i\}, \ldots, \{[\mathsf{x},m] \mid \mathsf{x} \in \mathcal{Z}_i\}\}$. The idea is to inductively incorporate all these string-equality selections $\varsigma^{=}_{\{[\mathsf{x},j]\mid\mathsf{x}\in\mathcal{Z}_i\}}$ one by one directly into the NFA M' by introducing variable references for the variables $\{[\mathsf{x},j] \mid \mathsf{x} \in \mathcal{Z}_i\}$ (note that this changes the accepted language from M into a ref-language). We sketch one individual step of this construction. Any ref-word accepted by (the current version of) M' contains (possibly zero) factors of the form ${}^{[\mathsf{x},j]}\!\triangleright v_{[\mathsf{x},j]} \triangleleft^{[\mathsf{x},j]}$ with $\mathsf{x} \in \mathcal{Z}_i$. It can be shown that $v_{[\mathsf{x},j]} = \gamma u_{[\mathsf{x},j]}\delta$, where $u_{[\mathsf{x},j]} \in \Sigma^*$ and γ and δ contain neither symbols from Σ nor variable references (the former is a consequence from the fact that $\mathcal{L}(M')$ is quasi-disjoint, the latter follows from the non-overlapping property of $\varsigma^{=}_{E}$). Moreover, these factors are pairwise non-overlapping. Now we want to check whether all these $u_{[\mathsf{x},j]}$ are the same, since this is required by $\varsigma^{=}_{\{[\mathsf{x},j]\mid\mathsf{x}\in\mathcal{Z}_i\}}$. This property is non-regular and can therefore not be checked directly. Instead, we only check whether the first ${}^{[\mathsf{x},j]}\!\triangleright \gamma u_{[\mathsf{x},j]}\delta \triangleleft^{[\mathsf{x},j]}$ that we encounter is such that we can potentially read the exact same word $u_{[\mathsf{x},j]}$ between all the remaining pairs of brackets ${}^{[\mathsf{y},j]}\!\triangleright \ldots \triangleleft^{[\mathsf{y},j]}$ for $\mathsf{y} \in \mathcal{Z}_i \setminus \{\mathsf{x}\}$. To this end, we guess the state pairs between which these factors will be read and then check, while reading $u_{[\mathsf{x},j]}$ from the input, whether we can also read $u_{[\mathsf{x},j]}$ between these states. For such words, we can then on-the-fly just read the variable reference $[\mathsf{x},j]$ instead $u_{[\mathsf{y},j]}$ when we reach the right states between which we have already checked that potentially $u_{[\mathsf{x},j]}$ could be read here. ◀

Theorem 6.5 says that if we allow an application of the span-fusion between the projection and the string-equality selections in the representation of the core-simplification lemma, then we can assume the regular spanner to be quasi-disjoint. Hence, we get the following:

▶ **Corollary 6.6.** $\mathsf{reg}\text{-}\mathfrak{S} = [\![\mathsf{RGX}]\!]^{\{\cup,\bowtie,\pi\}} = [\![\mathsf{RGX}]\!]^{\{\uplus,\pi\}}$ *and* $\mathsf{core}\text{-}\mathfrak{S} = [\![\mathsf{RGX}]\!]^{\{\cup,\bowtie,\pi,\varsigma^{=}\}} = [\![\mathsf{RGX}]\!]^{\{\uplus,\pi,\varsigma^{=}\}}$.

7 Conclusion

In this work, we introduced reference-bounded refl-spanners, a new fragment of core spanners. In terms of expressive power, this fragment is slightly less powerful than the class of core spanners, but has lower evaluation complexity (see Table 1, and note further that these upper bounds even hold for refl-spanners that are *not* necessarily reference-bounded). If we add the *span-fusion* – a natural binary operation on spanners – to reference-bounded refl-spanners (see Section 6) then they have the same expressive power as core spanners with *non-overlapping* string equality selections. This demonstrates that our formalism covers all aspects of core spanners except for the possibility of applying string equality selections on variables with overlapping spans. Moreover, since we achieve better complexities for refl-spanners compared to core spanners, this also shows that overlapping string equality selections are a source of complexity for core spanners.

From a conceptional point of view, our new angle was to treat the classical two-stage approach of core spanners, i. e., first producing the output table of a regular spanner and then filtering it by applying the string equality selections, as a single NFA. This is achieved by using ref-words in order to represent a document along with a span-tuple *that satisfies the string equality selections*, instead of just using subword-marked words to represent a document along with a span-tuple, which might not satisfy the string equality selections and therefore will be filtered out later in the second evaluation stage.

A question that is left open for further research is about constant delay enumeration for some fragment of core spanners strictly more powerful than the regular spanners. In this regard, we note that for efficient enumeration for (some fragments of) core spanners, we must overcome the general intractability of NonEmptiness (which we have for core spanners as well as for (reference bounded) refl-spanners). We believe that refl-spanners are a promising candidate for further restrictions that may lead to a fragment of core spanners with constant delay enumeration.

References

1 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *22nd International Conference on Database Theory, ICDT 2019, March 26–28, 2019, Lisbon, Portugal*, pages 22:1–22:19, 2019.

2 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9–11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466, 2016. doi:10.1109/FOCS.2016.56.

3 Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-correctness in information extraction. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 – July 5, 2019*, pages 149–163, 2019.

4 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12:1–12:51, 2015.

5 Henning Fernau, Markus L. Schmid, and Yngve Villanger. On the parameterised complexity of string morphism problems. *Theory of Computing Systems (ToCS)*, 59(1):24–51, 2016.

6 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10–15, 2018*, pages 165–177, 2018.

7 Dominik D. Freydenberger. A logic for document spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019. doi:10.1007/s00224-018-9874-1.

8 Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. *Theory Comput. Syst.*, 62(4):854–898, 2018. doi:10.1007/s00224-017-9770-0.

9 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10–15, 2018*, pages 137–149, 2018.

10 Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In *23rd International Conference on Database Theory, ICDT 2020, March 30 – April 2, 2020, Copenhagen, Denmark*, pages 11:1–11:21, 2020. doi:10.4230/LIPIcs.ICDT.2020.11.

11 Florin Manea and Markus L. Schmid. Matching patterns with variables. In *Combinatorics on Words – 12th International Conference, WORDS 2019, Loughborough, UK, September 9–13, 2019, Proceedings*, pages 1–27, 2019. doi:10.1007/978-3-030-28796-2_1.

12 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10–15, 2018*, pages 125–136, 2018.

13 Liat Peterfreund. *The Complexity of Relational Queries over Extractions from Text*. PhD thesis, Computer science department, Technion, 2019.

14 Liat Peterfreund. Grammars for document spanners. In *24th International Conference on Database Theory, ICDT 2021, March 23–26, 2021, Nicosia, Cyprus*, 2021. To appear. Extended version available at https://arxiv.org/abs/2003.06880.

15 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 – July 5, 2019.*, pages 320–334, 2019.

16 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 13:1–13:18, 2019.

17 Markus L. Schmid. Characterising REGEX languages by regular languages equipped with factor-referencing. *Information and Computation*, 249:1–17, 2016.

18 Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. *CoRR*, abs/2010.13442, 2020. arXiv:2010.13442.

Ranked Enumeration of Conjunctive Query Results

Shaleen Deep
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA

Paraschos Koutris
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA

Abstract

We study the problem of enumerating answers of Conjunctive Queries ranked according to a given ranking function. Our main contribution is a novel algorithm with small preprocessing time, logarithmic delay, and non-trivial space usage during execution. To allow for efficient enumeration, we exploit certain properties of ranking functions that frequently occur in practice. To this end, we introduce the notions of *decomposable* and *compatible* (w.r.t. a query decomposition) ranking functions, which allow for partial aggregation of tuple scores in order to efficiently enumerate the output. We complement the algorithmic results with lower bounds that justify why restrictions on the structure of ranking functions are necessary. Our results extend and improve upon a long line of work that has studied ranked enumeration from both a theoretical and practical perspective.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Query result enumeration, joins, ranking

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.5

Related Version *Full Version*: `https://arxiv.org/pdf/1902.02698.pdf` [17]

Funding This research was supported in part by National Science Foundation grants CRII-1850348 and III-1910014.

1 Introduction

For many data processing applications, enumerating query results according to an order given by a ranking function is a fundamental task. For example, [44, 10] consider a setting where users want to extract the top patterns from an edge-weighted graph, where the rank of each pattern is the sum of the weights of the edges in the pattern. Ranked enumeration also occurs in `SQL` queries with an `ORDER BY` clause [37, 26]. In the above scenarios, the user often wants to see the first k results in the query as quickly as possible, but the value of k may not be predetermined. Hence, it is critical to construct algorithms that can output the first tuple of the result as fast as possible, and then output the next tuple in the order with a very small *delay*. In this paper, we study the algorithmic problem of enumerating the result of a Conjunctive Query (CQ, for short) against a relational database where the tuples must be output in order given by a ranking function.

The simplest way to enumerate the output is to materialize the result $Q(D)$ and sort the tuples based on the score of each tuple. Although this approach is conceptually simple, it requires that $|Q(D)|$ tuples are materialized; moreover, the time from when the user submits the query to when she receives the first output tuples is $\Omega(|Q(D)| \cdot \log |Q(D)|)$. Further, the space and delay guarantees do not depend on the number of tuples that the user wants to actually see. More sophisticated approaches to this problem construct optimizers that exploit properties such as the monotonicity of the ranking function, allowing for join evaluation on a subset of the input relations (see [25] and references within). In spite of the significant progress, all of the known techniques suffer from large worst-case space requirements, no dependence on k, and provide no formal guarantees on the delay during enumeration, with the exception of a few cases where the ranking function is of a special form. Fagin et al. [21]

initiated a long line of study related to aggregation over *sorted lists*. However, [21] and subsequent works also suffer from the above mentioned limitations as we do not have the materialized output $Q(D)$ that can be used as sorted lists.

In this paper, we construct algorithms that remedy some of these issues. Our algorithms are divided into two phases: the *preprocessing phase*, where the system constructs a data structure that can be used later and the *enumeration phase*, when the results are generated. All of our algorithms aim to minimize the time of the preprocessing phase, and guarantee a *logarithmic delay* $O(\log |D|)$ during enumeration. Although we cannot hope to perform efficient ranked enumeration for an arbitrary ranking function, we show that our techniques apply for most ranking functions of practical interest, including lexicographic ordering, and sum (also product or max) of weights of input tuples among others.

▶ **Example 1.** Consider a weighted graph G, where an edge (a, b) with weight w is represented by the relation $R(a, b, w)$. Suppose that the user is interested in finding the (directed) paths of length 3 in the graph with the lowest score, where the score is a (weighted) sum of the weights of the edges. The user query in this case can be specified as: $Q(x, y, z, u, w_1, w_2, w_3,) = R(x, y, w_1), R(y, z, w_2), R(z, u, w_3)$ where the ranking of the output tuples is specified for example by the score $5w_1 + 2w_2 + 4w_3$. If the graph has N edges, the naïve algorithm that computes and ranks all tuples needs $\Omega(N^2 \log N)$ preprocessing time. We show that it is possible to design an algorithm with $O(N)$ preprocessing time, such that the delay during enumeration is $O(\log N)$. This algorithm outputs the first k tuples by materializing $O(N + k)$ data, even if the full output is much larger.

The problem of ranked enumeration for CQs has been studied both theoretically [28, 12, 36] and practically [44, 10, 5]. Theoretically, [28] establishes the tractability of enumerating answers in sorted order with polynomial delay (combined complexity), albeit with suboptimal space and delay factors for two classes of ranking functions. [44] presents an anytime enumeration algorithm restricted to acyclic queries on graphs that uses $\Theta(|Q(D)| + |D|)$ space in the worst case, has a $\Theta(|D|)$ delay guarantee, and supports only simple ranking functions. As we will see, both of these guarantees are suboptimal and can be improved upon.

Ranked enumeration has also been studied for the class of lexicographic orderings. In [3], the authors show that *free-connex acyclic CQs* can be enumerated in constant delay after only linear time preprocessing. Here, the lexicographic order is chosen by the algorithm and not the user. Factorized databases [5, 36] can also support constant delay ranked enumeration, but only when the lexicographic ordering agrees with the order of the query decomposition. In contrast, our results imply that we can achieve a logarithmic delay with the same preprocessing time for *any* lexicographic order.

Our Contribution. In this work, we show how to obtain logarithmic delay guarantees with small preprocessing time for ranking results of full (projection free) CQs. We summarize our technical contributions below:

1. Our main contribution (Theorem 12) is a novel algorithm that uses query decomposition techniques in conjunction with structure of the ranking function. The preprocessing phase sets up priority queues that maintain partial tuples at each node of the decomposition. During the enumeration phase, the algorithm materializes the output of the subquery formed by the subtree rooted at each node of the decomposition *on-the-fly*, in sorted order according to the ranking function. In order to define the rank of the partial tuples, we require that the ranking function can be *decomposed* with respect to the particular

decomposition at hand. Theorem 12 then shows that with $O(|D|^{\texttt{fhw}})$ preprocessing time, where $\texttt{fhw}$ is the *fractional hypertree width* of the decomposition, we can enumerate with delay $O(\log |D|)$. We then discuss how to apply our main result to commonly used classes of ranking functions. Our work thoroughly resolves an open problem stated at the Dagstuhl Seminar 19211 [8] on ranked enumeration (see Question 4.6).

2. We propose two extensions of Theorem 12 that improve the preprocessing time to $O(|D|^{\texttt{subw}})$, polynomial improvement over Theorem 12 where $\texttt{subw}$ is the *submodular width* of the query Q. The result is based on a simple but powerful corollary of the main result that can be applied to any full UCQ Q combined with the PANDA algorithm proposed by Abo Khamis et al. [1].
3. Finally, we show lower bounds (conditional and unconditional) for our algorithmic results. In particular, we show that subject to a popular conjecture, the logarithmic factor in delay cannot be removed. Additionally, we show that for two particular classes of ranking functions, we can characterize for which acyclic queries it is possible to achieve logarithmic delay with linear preprocessing time, and for which it is not.

2 Problem Setting

In this section we present the basic notions and terminology, and then discuss our framework.

2.1 Conjunctive Queries

In this paper we will focus on the class of *Conjunctive Queries (CQs)*, which are expressed as $Q(\mathbf{y}) = R_1(\mathbf{x}_1), R_2(\mathbf{x}_2), \ldots, R_n(\mathbf{x}_n)$ Here, the symbols $\mathbf{y}, \mathbf{x}_1, \ldots, \mathbf{x}_n$ are vectors that contain *variables* or *constants*, the atom $Q(\mathbf{y})$ is the *head* of the query, and the atoms $R_1(\mathbf{x}_1), R_2(\mathbf{x}_2), \ldots, R_n(\mathbf{x}_n)$ form the *body*. The variables in the head are a subset of the variables that appear in the body. A CQ is *full* if every variable in the body appears also in the head, and it is *boolean* if the head contains no variables, i.e. it is of the form $Q()$. We will typically use the symbols $x, y, z, \ldots$ to denote variables, and $a, b, c, \ldots$ to denote constants. We use $Q(D)$ to denote the result of the query Q over input database D. A *valuation* θ over a set V of variables is a total function that maps each variable $x \in V$ to a value $\theta(x) \in \mathbf{dom}$, where $\mathbf{dom}$ is a domain of constants. We will often use $\mathbf{dom}(x)$ to denote the constants that the valuations over variable x can take. It is implicitly understood that a valuation is the identity function on constants. If $U \subseteq V$, then $\theta[U]$ denotes the restriction of θ to U. A *Union of Conjunctive Queries* $\varphi = \bigcup_{i \in \{1,\ldots,\ell\}} \varphi_i$ is a set of CQs where $\text{head}(\varphi_{i_1}) = \text{head}(\varphi_{i_2})$ for all $1 \leq i_1, i_2 \leq \ell$. Semantically, $\varphi(D) = \bigcup_{i \in \{1,\ldots,\ell\}} \varphi_i(D)$. A UCQ is said to be full if each φ_i is full.

Natural Joins. If a CQ is full, has no constants and no repeated variables in the same atom, then we say it is a *natural join query*. For instance, the 3-path query $Q(x, y, z, w) = R(x, y), S(y, z), T(z, w)$ is a natural join query. A natural join can be represented equivalently as a *hypergraph* $\mathcal{H}_Q = (\mathcal{V}_Q, \mathcal{E}_Q)$, where $\mathcal{V}_Q$ is the set of variables, and for each hyperedge $F \in \mathcal{E}_Q$ there exists a relation R_F with variables F. We will write the join as $\bowtie_{F \in \mathcal{E}_Q} R_F$. We denote the size of relation R_F by $|R_F|$. Given two tuples t_1 and t_2 over a set of variables $\mathcal{V}_1$ and $\mathcal{V}_2$ where $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$, we will use $t_1 \circ t_2$ to denote the tuple formed over the variables $\mathcal{V}_1 \cup \mathcal{V}_2$. If $\mathcal{V}_1 \cap \mathcal{V}_2 \neq \emptyset$, then $t_1 \circ t_2$ will perform a join over the common variables.

Join Size Bounds. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph, and $S \subseteq \mathcal{V}$. A weight assignment $\mathbf{u} = (u_F)_{F \in \mathcal{E}}$ is called a *fractional edge cover* of S if (i) for every $F \in \mathcal{E}, u_F \geq 0$ and (ii) for every $x \in S, \sum_{F: x \in F} u_F \geq 1$. The *fractional edge cover number* of S, denoted by $\rho^*_{\mathcal{H}}(S)$ is the minimum of $\sum_{F \in \mathcal{E}} u_F$ over all fractional edge covers of S. We write $\rho^*(\mathcal{H}) = \rho^*_{\mathcal{H}}(\mathcal{V})$.

In a celebrated result, Atserias, Grohe and Marx [2] proved that for every fractional edge cover $\mathbf{u}$ of $\mathcal{V}$, the size of a natural join is bounded using the *AGM inequality*: $|\bowtie_{F \in \mathcal{E}} R_F| \leq \prod_{F \in \mathcal{E}} |R_F|^{u_F}$ The above bound is constructive [34, 33]: there exist worst-case algorithms that compute the join $\bowtie_{F \in \mathcal{E}} R_F$ in time $O(\prod_{F \in \mathcal{E}} |R_F|^{u_F})$ for every fractional edge cover $\mathbf{u}$ of $\mathcal{V}$.

Tree Decompositions. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph of a natural join query Q. A *tree decomposition* of $\mathcal{H}$ is a tuple $(\mathcal{T}, (\mathcal{B}_t)_{t \in V(\mathcal{T})})$ where $\mathcal{T}$ is a tree, and every $\mathcal{B}_t$ is a subset of $\mathcal{V}$, called the *bag* of t, such that

1. each edge in $\mathcal{E}$ is contained in some bag; and
2. for each variable $x \in \mathcal{V}$, the set of nodes $\{t \mid x \in \mathcal{B}_t\}$ is connected in $\mathcal{T}$.

Given a rooted tree decomposition, we use $\mathtt{p}(t)$ to denote the (unique) parent of node $t \in V(\mathcal{T})$. Then, we define $\mathtt{key}(t) = \mathcal{B}_t \cap \mathcal{B}_{\mathtt{p}(t)}$ to be the common variables that occur in the bag $\mathcal{B}_t$ and its parent, and $\mathtt{value}(t) = \mathcal{B}_t \setminus \mathtt{key}(t)$ the remaining variables of the bag. We also use $\mathcal{B}_t^{\preceq}$ to denote the union of all bags in the subtree rooted at t (including $\mathcal{B}_t$).

The *fractional hypertree width* of a decomposition is defined as $\max_{t \in V(\mathcal{T})} \rho^*(\mathcal{B}_t)$, where $\rho^*(\mathcal{B}_t)$ is the minimum fractional edge cover of the vertices in $\mathcal{B}_t$. The fractional hypertree width of a query Q, denoted $\mathtt{fhw}(Q)$, is the minimum fractional hypertree width among all tree decompositions of its hypergraph. We say that a query is *acyclic* if $\mathtt{fhw}(Q) = 1$. The *depth* of a rooted tree decomposition is the largest distance over all root to leaf paths in $\mathcal{T}$.

Computational Model. To measure the running time of our algorithms, we use the uniform-cost RAM model [24], where data values as well as pointers to databases are of constant size. Throughout the paper, all complexity results are with respect to data complexity (unless explicitly mentioned), where the query is assumed fixed.

2.2 Ranking Functions

Consider a natural join query Q and a database D. Our goal is to enumerate all the tuples of $Q(D)$ according to an order that is specified by a *ranking function*. In practice, this ordering could be specified, for instance, in the `ORDER BY` clause of a `SQL` query.

Formally, we assume a total order $\succeq$ of the valuations θ over the variables of Q. The total order is induced by a ranking function `rank` that maps each valuation θ to a number $\mathtt{rank}(\theta) \in \mathbb{R}$. In particular, for two valuations θ_1, θ_2, we have $\theta_1 \succeq \theta_2$ if and only if $\mathtt{rank}(\theta_1) \geq \mathtt{rank}(\theta_2)$. Throughout the paper, we will assume that `rank` is a computable function that takes times linear in the input size to the function . We present below two concrete examples of ranking functions.

▶ **Example 2.** For every constant $c \in \mathbf{dom}$, we associate a weight $w(c) \in \mathbb{R}$. Then, for each valuation θ, we can define $\mathtt{rank}(\theta) := \sum_{x \in \mathcal{V}} w(\theta(x))$. This ranking function sums the weights of each value in the tuple.

▶ **Example 3.** For every input tuple $t \in R_F$, we associate a weight $w_F(t) \in \mathbb{R}$. Then, for each valuation θ, we can define $\mathtt{rank}(\theta) = \sum_{F \in \mathcal{E}} w_F(\theta[x_F])$ where x_F is the set of variables in F. In this case, the ranking function sums the weights of each contributing input tuple to the output tuple t (we can extend the ranking function to all valuations by associating a weight of 0 to tuples that are not contained in a relation).

Decomposable Rankings. As we will see later, not all ranking functions are amenable to efficient evaluation. Intuitively, an arbitrary ranking function will require that we look across all tuples to even find the smallest or largest element. We next present several restrictions which are satisfied by ranking functions seen in practical settings.

▶ **Definition 4** (Decomposable Ranking). *Let* $\texttt{rank}$ *be a ranking function over* $\mathcal{V}$ *and* $S \subseteq \mathcal{V}$. *We say that* $\texttt{rank}$ *is* S-decomposable *if there exists a total order for all valuations over* S, *such that for every valuation* φ *over* $\mathcal{V} \setminus S$, *and any two valuations* θ_1, θ_2 *over* S *we have:*

$$\theta_1 \succeq \theta_2 \Rightarrow \texttt{rank}(\varphi \circ \theta_1) \geq \texttt{rank}(\varphi \circ \theta_2).$$

We say that a ranking function is *totally decomposable* if it is S-decomposable for every subset $S \subseteq \mathcal{V}$, and that it is *coordinate decomposable* if it is S-decomposable for any singleton set. Additionally, we say that it is *edge decomposable* for a query Q if it is S-decomposable for every set S that is a hyperedge in the query hypergraph. We point out here that totally decomposable functions are equivalent to monotonic orders as defined in [28].

▶ **Example 5.** The ranking function $\texttt{rank}(\theta) = \sum_{x \in \mathcal{V}} w(\theta(x))$ defined in Example 2 is totally decomposable, and hence also coordinate decomposable. Indeed, pick any set $S \subseteq \mathcal{V}$. We construct a total order on valuations θ over S by using the value $\sum_{x \in S} w(\theta(x))$. Now, consider valuations θ_1, θ_2 over S such that $\sum_{x \in S} w(\theta_1(x)) \geq \sum_{x \in S} w(\theta_2(x))$. Then, for any valuation φ over $\mathcal{V} \setminus S$ we have:

$$\begin{aligned}\texttt{rank}(\varphi \circ \theta_1) &= \sum_{x \in \mathcal{V} \setminus S} w(\varphi(x)) + \sum_{x \in S} w(\theta_1(x)) \geq \sum_{x \in \mathcal{V} \setminus S} w(\varphi(x)) + \sum_{x \in S} w(\theta_2(x)) \\ &= \texttt{rank}(\varphi \circ \theta_2)\end{aligned}$$

Next, we construct a function that is coordinate-decomposable but it is not totally decomposable. Consider the query

$$Q(x_1 \ldots, x_d, y_1, \ldots, y_d) = R(x_1, \ldots, x_d), S(y_1, \ldots, y_d)$$

where $\textbf{dom} = \{-1, 1\}$, and define $\texttt{rank}(\theta) := \sum_{i=1}^{d} \theta(x_i) \cdot \theta(y_i)$. This ranking function corresponds to taking the inner product of the input tuples if viewed as binary vectors. The total order for $\textbf{dom}$ is $-1 \prec 1$. It can be shown that for $d = 2$, the function is not $\{x_1, x_2\}$-decomposable. For instance, if we define $(1, 1) \succeq (1, -1)$, then inner product ranking function over x_1, x_2, y_1, y_2 for $\varphi = (1, -1)$ is $\texttt{rank}(1, 1, 1, -1) < \texttt{rank}(1, -1, 1, -1)$ but if we define $(1, -1) \succeq (1, 1)$, then for $\varphi = (-1, -1)$ we get $\texttt{rank}(1, -1, -1, -1) < \texttt{rank}(1, 1, -1, -1)$. This shows that there exists no total ordering over the valuations of variables $\{x_1, x_2\}$.

▶ **Definition 6.** *Let* $\texttt{rank}$ *be a ranking function over a set of variables* $\mathcal{V}$, *and* $S, T \subseteq \mathcal{V}$ *such that* $S \cap T = \emptyset$. *We say that* $\texttt{rank}$ *is* T-decomposable conditioned on S *if for every valuation* θ *over* S, *the function* $\texttt{rank}_\theta(\varphi) := \texttt{rank}(\theta \circ \varphi)$ *defined over* $\mathcal{V} \setminus S$ *is* T*-decomposable.*

The next lemma connects the notion of conditioned decomposability with decomposability.

▶ **Lemma 7.** *Let* $\texttt{rank}$ *be a ranking function over a set of variables* $\mathcal{V}$, *and* $T \subseteq \mathcal{V}$. *If* $\texttt{rank}$ *is* T*-decomposable, then it is also* T*-decomposable conditioned on* S *for any* $S \subseteq \mathcal{V} \setminus T$.

It is also easy to check that if a function is $(S \cup T)$-decomposable, then it is also T-decomposable conditioned on S.

▶ **Definition 8** (Compatible Ranking). *Let* $\mathcal{T}$ *be a rooted tree decomposition of hypergraph* $\mathcal{H}$ *of a natural join query. We say that a ranking function is* compatible with $\mathcal{T}$ *if for every node* t *it is* $(\mathcal{B}_t^{\prec} \setminus \texttt{key}(t))$*-decomposable conditioned on* $\texttt{key}(t)$.

▶ **Example 9.** Consider the join query $Q(x, y, z) = R(x, y), S(y, z)$, and the ranking function from Example 3, $\texttt{rank}(\theta) = w_R(\theta(x), \theta(y)) + w_S(\theta(y), \theta(z))$. This function is not $\{z\}$-decomposable, but it is $\{z\}$-decomposable conditioned on $\{y\}$.

Consider a decomposition of the hypergraph of Q that has two nodes: the root node r with $\mathcal{B}_r = \{x, y\}$, and its child t with $\mathcal{B}_t = \{y, z\}$. Since $\mathcal{B}_t^{\prec} = \{y, z\}$ and $\texttt{key}(t) = \{y\}$, the condition of compatibility holds for node t. Similarly, for the root node $\mathcal{B}_t^{\prec} = \{x, y, z\}$ and $\texttt{key}(t) = \{\}$, hence the condition is trivially true as well. Thus, the ranking function is compatible with the decomposition.

2.3 Problem Parameters

Given a natural join query Q and a database D, we want to enumerate the tuples of $Q(D)$ according to the order specified by `rank`. We will study this problem in the enumeration framework similar to that of [41], where an algorithm can be decomposed into two phases:

- a **preprocessing phase** that takes time T_p and computes a data structure of size S_p,
- an **enumeration phase** that outputs $Q(D)$ with no repetitions. The enumeration phase has full access to any data structures constructed in the preprocessing phase and can also use additional space of size S_e. The *delay* δ is defined as the maximum time to output any two consecutive tuples (and also the time to output the first tuple, and the time to notify that the enumeration has completed).

It is straightforward to perform ranked enumeration for any ranking function by computing $Q(D)$, storing the tuples in an ordered list, and finally enumerating by scanning the ordered list with constant delay. This simple strategy implies the following result.

▶ **Proposition 10.** *Let Q be a natural join query with hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. Let $\mathcal{T}$ be a tree decomposition with fractional hypertree-width $\texttt{fhw}$, and $\texttt{rank}$ be a ranking function. Then, for any input database D, we can preprocess D in time $T_p = O(\log D \cdot |D|^{\texttt{fhw}} + |Q(D)|)$ and space $S_p = O(|Q(D)|)$, such that for any k, we can enumerate the top-k results of $Q(D)$ with delay $\delta = O(1)$ and space $S_e = O(1)$*

The drawback of Proposition 10 is that the user will have to wait $\Omega(|Q(D)| \cdot \log |Q(D)|)$ time to even obtain the first tuple in the output. Moreover, even when we are interested in a few tuples, the whole output result will have to be materialized. Instead, we want to design algorithms that minimize the preprocessing time and space, while guaranteeing a small delay δ. Interestingly, as we will see in Section 5, the above result is essentially the best we can do if the ranking function is completely arbitrary; thus, we need to consider reasonable restrictions of `rank`.

To see what it is possible to achieve in this framework, it will be useful to keep in mind what we can do in the case where there is no ordering of the output.

▶ **Theorem 11** (due to [36]). *Let Q be a natural join query with hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. Let $\mathcal{T}$ be a tree decomposition with fractional hypertree-width $\texttt{fhw}$. Then, for any input database D, we can pre-process D in time $T_p = O(|D|^{\texttt{fhw}})$ and space $S_p = O(|D|^{\texttt{fhw}})$ such that we can enumerate the results of $Q(D)$ with delay $\delta = O(1)$ and space $S_e = O(1)$*

For acyclic queries, $\texttt{fhw} = 1$, and hence the preprocessing phase takes only linear time and space in the size of the input.

3 Main Result

In this section, we present our first main result.

▶ **Theorem 12** (Main Theorem). *Let Q be a natural join query with hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. Let $\mathcal{T}$ be a fixed tree decomposition with fractional hypertree-width $\mathtt{fhw}$, and $\mathtt{rank}$ be a ranking function that is compatible with $\mathcal{T}$. Then, for any database D, we can preprocess D with*

$$T_p = O(|D|^{\mathtt{fhw}}) \qquad S_p = O(|D|^{\mathtt{fhw}})$$

such that for any k, we can enumerate the top-k tuples of $Q(D)$ with

$$\textit{delay } \delta = O(\log |D|) \qquad \textit{space } S_e = O(\min\{k, |Q(D)|\})$$

In the above theorem, the preprocessing step is independent of the value of k: we perform exactly the same preprocessing if the user only wants to obtain the first tuple, or all tuples in the result. However, if the user decides to stop after having obtained the first k results, the space used during enumeration will be bound by $O(k)$. We should also note that all of our algorithms work in the case where the ordering of the tuples/valuations is instead expressed through a `comparable` function that, given two valuations, returns the largest one.

It is instructive to compare Theorem 12 with Theorem 11, where no ranking is used when enumerating the results. There are two major differences. First, the delay δ has an additional logarithmic factor. As we will discuss later in Section 5, this logarithmic factor is a result of doing ranked enumeration, and it is most likely unavoidable. The second difference is that the space S_e used during enumeration blows up from constant $O(1)$ to $O(|Q(D)|)$ in the worst case (when all results are enumerated).

In the remainder of this section, we will present a few applications of Theorem 12, and then sketch the construction for the proof of the theorem.

3.1 Applications

We show here how to apply Theorem 12 to obtain algorithms for different ranking functions.

Vertex-Based Ranking. A vertex-based ranking function over $\mathcal{V}$ is of the form: $\mathtt{rank}(\theta) := \bigoplus_{x \in \mathcal{V}} f_x(\theta(x))$ where f_x maps values from **dom** to some set $U \subseteq \mathbb{R}$, and $\langle U, \oplus \rangle$ forms a *commutative monoid*. Recall that this means that $\oplus$ is a binary operator that is commutative, associative, and has an identity element in U. We say that the function is monotone if $a \geq b$ implies that $a \oplus c \geq b \oplus c$ for every c. Such examples are $\langle \mathbb{R}, + \rangle$, $\langle \mathbb{R}, * \rangle$, and $\langle U, max \rangle$, where U is bounded.

▶ **Lemma 13.** *Let $\mathtt{rank}$ be a monotone vertex-based ranking function over $\mathcal{V}$. Then, $\mathtt{rank}$ is totally decomposable, and hence compatible with any tree decomposition of a hypergraph with vertices $\mathcal{V}$.*

Tuple-Based Ranking. Given a query hypergraph $\mathcal{H}$, a tuple-based ranking function assigns for every valuation θ over the variables x_F of relation R_F a weight $w_F(\theta) \in U \subseteq \mathbb{R}$. Then, it takes the following form: $\mathtt{rank}(\theta) := \bigoplus_{F \in \mathcal{E}} w_F(\theta[x_F])$ where $\langle U, \oplus \rangle$ forms a *commutative monoid*. In other words, a tuple-based ranking function assigns a weight to each input tuple, and then combines the weights through $\oplus$.

▶ **Lemma 14.** *Let $\mathtt{rank}$ be a monotone tuple-based ranking function over $\mathcal{V}$. Then, $\mathtt{rank}$ is compatible with any tree decomposition of a hypergraph with vertices $\mathcal{V}$.*

Since both monotone tuple-based and vertex-based ranking functions are compatible with any tree decomposition we choose, the following result is immediate.

▶ **Proposition 15.** *Let Q be a natural join query with optimal fractional hypertree-width* $\mathtt{fhw}$*. Let* $\mathtt{rank}$ *be a ranking function that can be either (i) monotone vertex-based, (ii) monotone tuple-based. Then, for any input D, we can pre-process D in time $T_p = O(|D|^{\mathtt{fhw}})$ and space $S_p = O(|D|^{\mathtt{fhw}})$ such that for any k, we can enumerate the top-k results of $Q(D)$ with $\delta = O(\log |D|)$ and $S_e = O(\min\{k, |Q(D)|\})$*

For instance, if the query is acyclic, hence $\mathtt{fhw} = 1$, the above theorem gives an algorithm with linear preprocessing time $O(|D|)$ and $O(\log |D|)$ delay.

Lexicographic Ranking. A typical ordering of the output valuations is according to a *lexicographic order*. In this case, each $\mathbf{dom}(x)$ is equipped with a total order. If $\mathcal{V} = \{x_1, \ldots, x_k\}$, a lexicographic order $\langle x_{i_1}, \ldots, x_{i_\ell}\rangle$ for $\ell \leq k$ means that two valuations θ_1, θ_2 are first ranked on x_{i_1}, and if they have the same rank on x_{i_1}, then they are ranked on x_{i_2}, and so on. This ordering can be naturally encoded by first taking a function $f_x : \mathbf{dom}(x) \to \mathbb{R}$ that captures the total order for variable x, and then defining $\mathtt{rank}(\theta) := \sum_x w_x f_x(\theta(x))$, where w_x are appropriately chosen constants. Since this ranking function is a monotone vertex-based ranking, Proposition 15 applies here as well.

We should note here that lexicographic ordering has been previously considered in the context of factorized databases.

▶ **Proposition 16** (due to [36, 5])**.** *Let Q be a natural join query with hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, and $\langle x_{i_1}, \ldots, x_{i_\ell}\rangle$ a lexicographic ordering of the variables in $\mathcal{V}$.*

Let $\mathcal{T}$ be a tree decomposition with fractional hypertree-width $\mathtt{fhw\text{-}lex}$ *such that $\langle x_{i_1}, \ldots, x_{i_\ell}\rangle$ forms a prefix in the topological ordering of the variables in the decomposition. Then, for any input database D, we can pre-process D with $T_p = O(|D|^{\mathtt{fhw\text{-}lex}})$ and $S_p = O(|D|^{\mathtt{fhw\text{-}lex}})$ such that results of $Q(D)$ can be enumerated with delay $\delta = O(1)$ and space $S_e = O(1)$.*

In other words, if the lexicographic order "agrees" with the tree decomposition (in the sense that whenever x_i is before x_j in the lexicographic order, x_j can never be in a bag higher than the bag where x_i is), then it is possible to get an even better result than Theorem 12, by achieving constant delay $O(1)$, and constant space S_e. However, given a tree decomposition, Theorem 12 applies for any lexicographic ordering - in contrast to Proposition 16. As an example, consider the join query $Q(x, y, z) = R(x, y), S(y, z)$ and the lexicographic ordering $\langle z, x, y\rangle$. Since $\mathtt{fhw} = 1$, our result implies that we can achieve $O(|D|)$ time preprocessing with delay $O(\log |D|)$. On the other hand, the optimal width of a tree decomposition that agrees with $\langle z, x, y\rangle$ is $\mathtt{fhw\text{-}lex} = 2$; hence, Proposition 16 implies $O(|D|^2)$ preprocessing time and space. Thus, variable orderings in a decomposition fail to capture the additional challenge of user chosen lexicographic orderings. It is also not clear whether further restrictions on variable orderings in Proposition 16 are sufficient to capture ordered enumeration for other ranking functions (such as sum).

Bounded Ranking. A ranking function is *c-bounded* if there exists a subset $S \subseteq \mathcal{V}$ of size $|S| = c$, such that the value of $\mathtt{rank}$ depends only on the variables from S. A c-bounded ranking is related to c-determined ranking functions [28]: c-determined implies c-bounded, but not vice versa. For c-bounded ranking functions, we can show the following result:

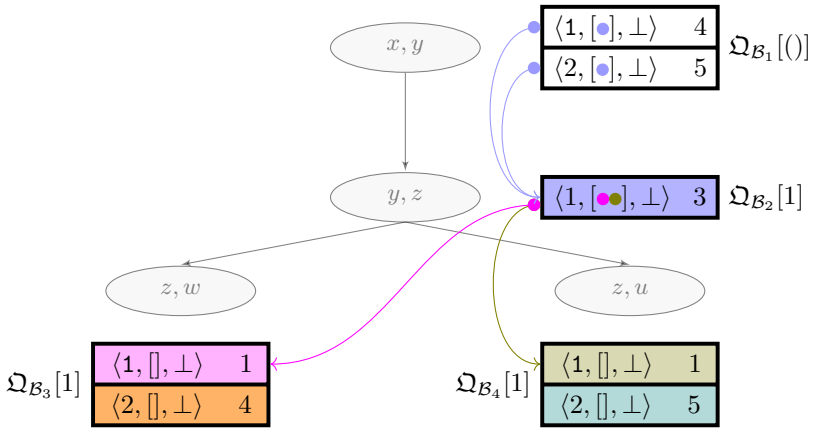
▶ **Proposition 17.** *Let Q be a natural join query with optimal fractional hypertree-width $\mathtt{fhw}$. If $\mathtt{rank}$ is a c-bounded ranking function, then for any input D, we can pre-process D in time $T_p = O(|D|^{\mathtt{fhw}+c})$ and space $S_p = O(|D|^{\mathtt{fhw}+c})$ such that for any k, we can enumerate the top-k results of $Q(D)$ with $\delta = O(\log |D|)$ and $S_e = O(\min\{k, |Q(D)|\})$*

3.2 The Algorithm for the Main Theorem

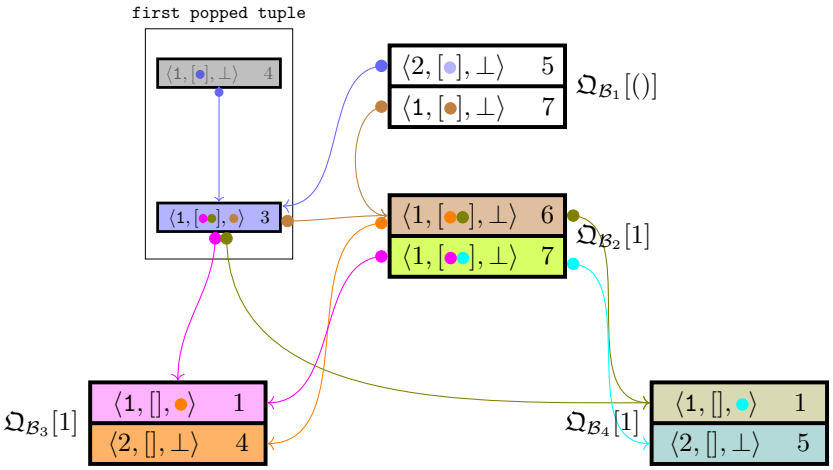
At a high level, each node t in the tree decomposition will materialize in an incremental fashion all valuations over $\mathcal{B}_t^{\prec}$ that satisfy the query that corresponds to the subtree rooted at t. We do not store explicitly each valuation θ over $\mathcal{B}_t^{\prec}$ at every node t, but instead we use a simple recursive structure $C(v)$ that we call a *cell*. If t is a leaf, then $C(\theta) = \langle \theta, [], \perp \rangle$, where $\perp$ is used to denote a null pointer. Otherwise, suppose that t has n children $t_1, \ldots, t_n$. Then, $C(\theta) = \langle \theta[\mathcal{B}_t], [p_1, \ldots, p_n], q \rangle$, where p_i is a pointer to the cell $C(\theta[\mathcal{B}_{t_i}^{\prec}])$ stored at node t_i, and q is a pointer to a cell stored at node t (intuitively representing the "next" valuation in the order). It is easy to see that, given a cell $C(\theta)$, one can reconstruct θ in constant time (dependent only on the query). Additionally, each node t maintains one hash map $\mathfrak{Q}_t$, which maps each valuation u over $\mathtt{key}(\mathcal{B}_t)$ to a *priority queue* $\mathfrak{Q}_t[u]$. The elements of $\mathfrak{Q}_t$ are cells $C(\theta)$, where θ is a valuation over $\mathcal{B}_t^{\prec}$ such that $u = \theta[\mathtt{key}(\mathcal{B}_t)]$. The priority queues will be the data structure that performs the comparison and ordering between different tuples. We will use an implementation of a priority queue (e.g., a Fibonacci heap [13]) with the following properties: (i) we can insert an element in constant time $O(1)$, (ii) we can obtain the min element (top) in time $O(1)$, and (iii) we can delete the min element (pop) in time $O(\log n)$.

Notice that it is not straightforward to rank the cells according to the valuations, since the ranking function is defined over all variables $\mathcal{V}$. However, here we can use the fact that the ranking function is compatible with the decomposition at hand. Indeed, given a fixed valuation u over $\mathtt{key}(\mathcal{B}_t)$, we will order the valuations θ over $\mathcal{B}_t^{\prec}$ that agree with u according to the score: $\mathtt{rank}(v_t^\star \circ \theta)$ where $v_t^\star$ is a valuation over $\mathcal{V} \setminus \mathcal{B}_t^{\prec}$ chosen according to the definition of decomposability. The key intuition is that the compatibility of the ranking function with the decomposition implies that the ordering of the tuples in the priority queue $\mathfrak{Q}_t[u]$ will not change if we replace $v_t^\star$ with any other valuation. Thus, the comparator can use $v_t^\star$ to calculate the score which is used by the priority queue internally. We next discuss the *preprocessing* and *enumeration* phase of the algorithm.

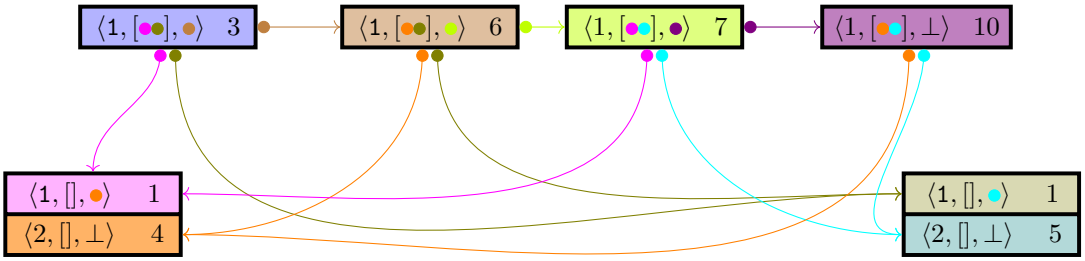
Preprocessing. Algorithm 1 consists of two steps. The first step works exactly as in the case where there is no ranking function: each bag $\mathcal{B}_t$ is computed and materialized, and then we apply a full reducer pass to remove all tuples from the materialized bags that will not join in the final result. The second step initializes the hash map with the priority queues for every bag in the tree. We traverse the decomposition in a bottom up fashion (post-order traversal), and do the following. For a leaf node t, notice that the algorithm does not enter the loop in line 10, so each valuation θ over $\mathcal{B}_t$ is added to the corresponding queue as the triple $\langle \theta, [], \perp \rangle$. For each non-leaf node t, we take each valuation v over $\mathcal{B}_t$ and form a valuation (in the form of a cell) over $\mathcal{B}_t^{\prec}$ by using the valuations with the largest rank from its children (we do this by accessing the top of the corresponding queues in line 10). The cell is then added to the corresponding priority queue of the bag. Observe that the root node r has only one priority queue, since $\mathtt{key}(r) = \{\}$.



(a) Priority queue state (mirroring the decomposition) after preprocessing phase.



(b) Priority queue state after one iteration of loop in procedure `ENUM()`.



(c) The materialized output stored at subtree rooted at $\mathcal{B}_2$ after enumeration is complete.

Figure 1 Preprocessing and enumeration phase for Example 1. Each memory location is shown with a different color. Pointers in cells are denoted using ● which means that the it points to a memory location with the corresponding color (shown using pointed arrows). Root bag priority queue cells are not color coded as nobody points to them.

Algorithm 1 Preprocessing Phase.

```
1  foreach t ∈ V(T) do
2  |  materialize the bag B_t
3  full reducer pass on materialized bags in T
4  forall t ∈ V(T) in post-order traversal do
5  |  foreach valuation θ in bag B_t do
6  |  |  u ← θ[key(B_t)]
7  |  |  if Ω_t[u] is NULL then
8  |  |  |  Ω_t[u] ← new priority queue
9  |  |  ℓ ← []
   |  |  /* ℓ is a list of pointers                                   */
10 |  |  foreach child s of t do
11 |  |  |  ℓ.INSERT(Ω_s[θ[key(B_s)]].TOP())
12 |  |  Ω_t[u].INSERT(⟨θ, ℓ, ⊥⟩) /* ranking function uses θ, ℓ, v*_t to calculate score used
   |  |     by priority queue                                         */
```

Algorithm 2 Enumeration Phase.

```
1  procedure ENUM()
2  |  while Ω_r[()] is not empty do
3  |  |  output Ω_r[()].TOP()
4  |  |  TOPDOWN(Ω_r[()].TOP(), r)
5  procedure TOPDOWN(c, t)
6  |  /* c = ⟨θ, [p_1, ..., p_k], next⟩ */
7  |  u ← θ[key(B_t)]
8  |  if next = ⊥ then
9  |  |  Ω_t[u].POP()
10 |  |  foreach child t_i of t do
11 |  |  |  p'_i ← TOPDOWN(*p_i, t_i)
12 |  |  |  if p'_i ≠ ⊥ then
13 |  |  |  |  Ω_t[u].INSERT(⟨θ, [p_1, ..., p'_i, ... p_k], ⊥⟩) /* insert new candidate(s) */
14 |  |  if t is not the root then
15 |  |  |  next ← Ω_t[u].TOP()
16 |  |  return next
```

▶ **Example 18.** As a running example, we consider the natural join query $Q(x, y, z, w) = R_1(x, y), R_2(y, z), R_3(z, w), R_4(z, u)$ where the ranking function is the sum of the weights of each input tuple. Consider the following instance D and decomposition $\mathcal{T}$ for our running example.

R_1

id	$\mathbf{w_1}$	**x**	**y**
1	1	1	1
2	2	2	1

R_2

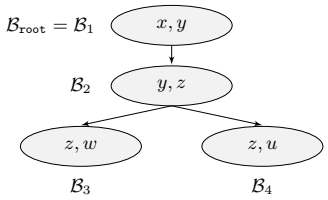
id	$\mathbf{w_2}$	**y**	**z**
1	1	1	1
2	1	3	1

R_3

id	$\mathbf{w_3}$	**z**	**w**
1	1	1	1
2	4	1	2

R_4

id	$\mathbf{w_3}$	$\mathbf{z}$	$\mathbf{u}$
1	1	1	1
2	5	1	2



For the instance shown above and the query decomposition that we have fixed, relation R_i covers bag $\mathcal{B}_i, i \in [4]$. Each relation has size $N = 2$. Since the relations are already materialized, we only need to perform a full reducer pass, which can be done in linear time. This step removes tuple $(3, 1)$ from relation R_2 as it does not join with any tuple in R_1.

Figure 1a shows the state of priority queues after the pre-processing step. For convenience, θ in each cell $\langle \theta, [p_1, \ldots, p_k], \mathsf{next} \rangle$ is shown using the primary key of the tuple and pointers p_i and next are shown using colored dots ● representing the memory location it points to. The cell in a memory location is followed by the partial aggregated score of the tuple formed by creating the tuple from the pointers in the cell recursively. For instance, the score of the tuple formed by joining $(y = 1, z = 1) \in R_2$ with $(z = 1, w = 1)$ from R_3 and $(z = 1, u = 1)$ in R_4 is $1 + 1 + 1 = 3$ (shown as ⟨1, [●●], ⊥⟩ 3 in the figure). Each cell in every priority queue points to the top element of the priority queue of child nodes that are joinable. Note that since both tuples in R_1 join with the sole tuple from R_2, they point to the same cell.

Enumeration. Algorithm 2 presents the algorithm for the enumeration phase. The heart of the algorithm is the procedure `TOPDOWN`(c, t). The key idea of the procedure is that whenever we want to output a new tuple, we can simply obtain it from the top of the priority queue in the root node (node r is the root node of the tree decomposition). Once we do that, we need to update the priority queue by popping the top, and inserting (if necessary) new valuations in the priority queue. This will be recursively propagated in the tree until it reaches the leaf nodes. Observe that once the new candidates have been inserted, the next pointer of cell c is updated by pointing to the topmost element in the priority queue. This chaining materializes the answers for the particular bag that can be reused.

▶ **Example 19.** Figure 1b shows the state of the data structure after one iteration in `ENUM()`. The first answer returned to the user is the topmost tuple from $\mathfrak{Q}_{\mathcal{B}_1}[()]$ (shown in top left of the figure). Cell ⟨1, [ ], ⊥⟩ 4 is popped from $\mathfrak{Q}_{\mathcal{B}_1}[()]$ (after satisfying if condition on line 8 as next is ⊥). Since nothing is pointing to this cell, it is garbage collected (denoted by greying out the cell). We recursively call `TOPDOWN` for child node $\mathcal{B}_2$ and cell ⟨1, [●●], ⊥⟩ 3 . The next for this cell is also ⊥ and we pop it from $\mathfrak{Q}_{\mathcal{B}_2}[1]$. At this point, $\mathfrak{Q}_{\mathcal{B}_2}[1]$ is empty. The next recursive call is for $\mathcal{B}_3$ with ⟨1, [], ⊥⟩ 1 . The least ranked tuple but larger than ⟨1, [], ⊥⟩ 1 in $\mathfrak{Q}_{\mathcal{B}_3}[1]$ is the cell at address ● . Thus, next for ⟨1, [], ⊥⟩ 1 is updated to ● and cell at ● is returned which leads to creation and insertion of ⟨1, [●●], ⊥⟩ 6 cell in $\mathfrak{Q}_{\mathcal{B}_2}[1]$. Similarly, we get the other cell in $\mathfrak{Q}_{\mathcal{B}_2}[1]$ by recursive call for $\mathcal{B}_4$. After both the calls are over for node $\mathcal{B}_2$, the topmost cell at $\mathfrak{Q}_{\mathcal{B}_2}[1]$ is ● ,which is set as the next for ⟨1, [●●], ⊥⟩ 3 (changing into ⟨1, [●●], ●⟩ 3), terminating one full iteration. ⟨1, [●●], ●⟩ 3 is not garbage collected as ⟨2, [●], ⊥⟩ 5 is pointing to it.

Let us now look at the second iteration of `ENUM()`. The tuple returned is top element of $Q_{\mathcal{B}_1}[()]$ which is ⟨2, [●], ⊥⟩ 5 . However, the function `TOPDOWN()` with ⟨2, [●], ⊥⟩ 5 does not recursively go all the way down to leaf nodes. Since ⟨1, [●●], ●⟩ 3 already has next populated, we insert ⟨2, [●], ⊥⟩ 5 in $Q_{\mathcal{B}_1}[()]$ completing the iteration. This demonstrates the benefit of materializing ranked answers at each node in the tree. As the enumeration continues, we are

materializing the output of each subtree on-the-fly that can be reused by other tuples in the root bag. Figure 1c shows the eventual sequence of pointers at node $\mathcal{B}_2$ which is the ranked materialized output of the subtree rooted at $\mathcal{B}_2$. $\boxed{\langle 2, [\circledast], \perp\rangle \quad 5}$ is garbage collected.

4 Extensions

In this section, we describe two extensions of Theorem 12 and how it can be used to further improve the main result.

4.1 Ranked Enumeration of UCQs

We begin by discussing how ranked enumeration of full UCQs can be done. The first observation is that given a full UCQ $\varphi = \varphi_1 \cup \ldots \varphi_\ell$, if the ranked enumeration of each φ_i can be performed efficiently, then we can perform ranked enumeration for the union of query results. This can be achieved by applying Theorem 12 to each φ_i and introducing another priority queue that compares the score of the answer tuples of each φ_i, pops the smallest result, and fetches the next smallest tuple from the data structure of φ_i accordingly. Although each $\varphi_i(D)$ does not contain duplicates, it may be the case that the same tuple is generated by multiple φ_i. Thus, we need to introduce a mechanism to ensure that all tuples with the same weight are enumerated in a specific order. Fortunately, this is easy to accomplish by modifying Algorithm 2 to enumerate all tuples with the same score in lexicographic increasing order. This ensures that tuples from each φ_i also arrive in the same order. Since each φ_i is enumerable in ranked order with delay $O(\log |D|)$ and the overhead of the priority queue is $O(\ell)$ (priority queue contains at most one tuple from each φ_i), the total delay guarantee is bounded by $O(\ell \cdot \log |D|) = O(\log |D|)$ as the query size is a constant. The space usage is determined by the largest fractional hypertree-width across all decompositions of subqueries in φ. This immediately leads to the following corollary of the main result.

▶ **Corollary 20.** *Let $\varphi = \varphi_1 \cup \ldots \varphi_\ell$ be a full UCQ. Let* **fhw** *denote the fractional hypertree-width of all decompositions across all CQs φ_i, and* **rank** *be a ranking function that is compatible with the decomposition of each φ_i. Then, for any input database D, we can pre-process D in time and space,*

$$T_p = O(|D|^{fhw}) \qquad S_p = O(|D|^{fhw})$$

such that for any k, we can enumerate the top-k tuples of $\varphi(D)$ with

$$\text{delay } \delta = O(\log |D|) \qquad \text{space } S_e = O(\min\{k, |\varphi(D)|\})$$

4.2 Improving The Main Result

Although Corollary 20 is a straightforward extension of Theorem 12, it is powerful enough to improve the pre-processing time and space of Theorem 12 by using Corollary 20 in conjunction with *data-dependent* tree decompositions. It is well known that the query result for any CQ can be answered in time $O(|D|^{\texttt{fhw}} + |Q(D)|)$ time and this is asymptotically tight [2]. However, there exists another notion of width known as the *submodular width* (denoted `subw`) [31]. It is also known that for any CQ, it holds that $\texttt{subw} \leq \texttt{fhw}$. Recent work by Abo Khamis et al. [1] presented an elegant algorithm called `PANDA` that constructs multiple decompositions by partitioning the input database to minimize the intermediate join size result. `PANDA` computes the output of any full CQ in time $O(|D|^{\texttt{subw}} \cdot \log |D| + |Q(D)|)$. In other words, `PANDA` takes a CQ query Q and a database D as input and produces multiple

tree decompositions in time $O(|D|^{\texttt{subw}} \cdot \log |D|)$ such that each answer tuple is generated by at least one decomposition. The number of decompositions depends only on size of the query and not on D. Thus, when the query size is a constant, the number of decompositions constructed is also a constant. We can now apply Corollary 20 by setting φ_i as the tree decompositions produced by `PANDA` to get the following result. [17] describes the details of the enumeration algorithm and the tie-breaking comparison function.

▶ **Theorem 21.** *Let φ be a natural join query with hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, submodular width **subw**, and **rank** be a ranking function that is compatible with each tree decomposition of φ. Then, for any input database D, we can pre-process D in time and space,*

$$T_p = O(|D|^{subw} \cdot \log |D|) \qquad S_p = O(|D|^{subw})$$

such that for any k, we can enumerate the top-k tuples of $\varphi(D)$ with

$$\text{delay } \delta = O(\log |D|) \qquad \text{space } S_e = O(\min\{k, |\varphi(D)|\})$$

5 Lower Bounds

In this section, we provide evidence for the near optimality of our results.

5.1 The Choice of Ranking Function

We first consider the impact of the ranking function on the performance of ranked enumeration. We start with a simple observation that deals with the case where `rank` has no structure, and can be accessed only through a blackbox that, given a tuple/valuation, returns its score: we call this a *blackbox* [1] ranking function. Note that all of our algorithms work under the blackbox assumption.

▶ **Proposition 22.** *Let Q be a natural join query, and **rank** a blackbox ranking function. Then, any enumeration algorithm on a database D needs $\Omega(|Q(D)|)$ calls to **rank**– and worst case $\Omega(|D|^{\rho^*})$ calls – in order to output the smallest tuple.*

Indeed, if the algorithm does not examine the rank of an output tuple, then we can always assign a value to the ranking function such that the tuple is the smallest one. Hence, in the case where there is no restriction on the ranking function, the simple result in Proposition 10 that materializes and sorts the output is essentially optimal. Thus, it is necessary to exploit properties of the ranking function in order to construct better algorithms. Unfortunately, even for natural restrictions of ranking functions, it is not possible to do much better than the $|D|^{\rho^*}$ bound for certain queries.

Such a natural restriction is that of coordinate decomposable functions, where we can show the following lower bound result:

▶ **Lemma 23.** *Consider the query $Q(x_1, y_1, x_2, y_2) = R(x_1, y_1), S(x_2, y_2)$ and let **rank** be a blackbox coordinate decomposable ranking function. Then, there exists an instance of size N such that the time required to find the smallest tuple is $\Omega(N^2)$.*

Lemma 23 shows that for coordinate decomposable functions, there exist queries where obtaining constant (or almost constant) delay requires the algorithm to spend superlinear time during the preprocessing step. Given this result, the immediate question is to see whether we can extend the lower bound to other CQs.

[1] Blackbox implies that the score $\texttt{rank}(\theta)$ is revealed only upon querying the function.

Dichotomy for coordinate decomposable. We first show a dichotomy result for coordinate decomposable functions.

▶ **Theorem 24.** *Consider a full acyclic query Q and a coordinate decomposable blackbox ranking function. There exists an algorithm that enumerates the result of Q in ranked order with $O(\log |D|)$ delay guarantee and $T_p = O(|D|)$ preprocessing time if and only if there are no atoms R and S in Q such that $\mathsf{vars}(R) \setminus \mathsf{vars}(S) \geq 2$ and $\mathsf{vars}(S) \setminus \mathsf{vars}(R) \geq 2$.*

For example, the query $Q(x, y, z) = R(x, y), S(y, z)$ satisfies the condition of Theorem 24, while the Cartesian product query defined in Lemma 23 does not.

Dichotomy for edge decomposable. We will show a dichotomy result for edge decomposable ranking functions: these are functions that are S-decomposable for any S that is a hyperedge in the query hypergraph. Before we present the result, we need to formally define the notion of path and diameter in a hypergraph.

▶ **Definition 25.** *Given a connected hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, a path P in $\mathcal{H}$ from vertex x_1 to x_{s+1} is a vertex-edge alternate set $x_1 E_1 x_2 E_2 \dots x_s E_s x_{s+1}$ such that $\{x_i, x_{i+1}\} \subseteq E_i (i \in [s])$ and $x_i \neq x_j, E_i \neq E_j$ for $i \neq j$. Here, s is the length of the path P. The distance between any two vertices u and v, denoted $d(u, v)$, is the length of the shortest path connecting u and v. The* diameter *of a hypergraph, $\mathsf{dia}(\mathcal{H})$, is the maximum distance between all pairs of vertices.*

▶ **Theorem 26.** *Consider a full connected acyclic join query Q and a blackbox edge decomposable ranking function. Then, there exists an algorithm that enumerates the result of Q in ranked order with $O(\log |D|)$ delay and $T_p = O(|D|)$ preprocessing time if and only if $\mathsf{dia}(Q) \leq 3$.*

For example, $Q(x, y, z, w) = R(x, y), S(y, z), T(z, w)$ has diameter 3, and thus we can enumerate the result with linear preprocessing time and logarithmic delay for any edge decomposable ranking function. On the other hand, for the 4-path query $Q(x, y, z, w, t) = R(x, y), S(y, z), T(z, w), U(w, t)$, it is not possible to achieve this.

When the query is not connected, the characterization must be slightly modified: an acyclic query can be enumerated with $O(\log |D|)$ delay and $T_p = O(|D|)$ if and only if each connected subquery has diameter at most 3.

5.2 Beyond Logarithmic Delay

Next, we examine whether the logarithmic factor that we obtain in the delay of Theorem 12 can be removed for ranked enumeration. In other words, is it possible to achieve constant delay enumeration while keeping the preprocessing time small, even for simple ranking functions? To reason about this, we need to describe the $X + Y$ *sorting* problem.

Given two lists of n numbers, $X = \langle x_1, x_2, \dots, x_n \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$, we want to enumerate all n^2 pairs (x_i, y_j) in ascending order of their sum $x_i + y_j$. This classic problem has a trivial $O(n^2 \log n)$ algorithm that materializes all n^2 pairs and sorts them. However, it remains an open problem whether the pairs can be enumerated faster in the RAM model. Fredman [22] showed that $O(n^2)$ comparisons suffice in the nonuniform linear decision tree model, but it remains open whether this can be converted into an $O(n^2)$-time algorithm in the real RAM model. Steiger and Streinu [43] gave a simple algorithm that takes $O(n^2 \log n)$ time while using only $O(n^2)$ comparisons.

▶ **Conjecture 27** ([9, 18]). *$X + Y$ sorting does not admit an $O(n^2)$ time algorithm.*

In our setting, $X + Y$ sorting can be expressed as enumerating the output of the cartesian product $Q(x, y) = R(x), S(y)$, where relations R and S correspond to the sets X and Y respectively. The ranking function is $\texttt{rank}(x, y) = x + y$. Conjecture 27 implies that it is not possible to achieve constant delay for the cartesian product query and the sum ranking function; otherwise, a full enumeration would produce a sorted order in time $O(n^2)$.

6 Related Work

Top-k ranked enumeration of join queries has been studied extensively by the database community for both certain [29, 37, 26, 30] and uncertain databases [38, 45]. Most of these works exploit the monotonicity property of scoring functions, building offline indexes and integrate the function into the cost model of the query optimizer in order to bound the number of operations required per answer tuple. We refer the reader to [25] for a comprehensive survey of top-k processing techniques. More recent work [10, 23] has focused on enumerating *twig-pattern* queries over graphs. Our work departs from this line of work in two aspects: *(i)* use of novel techniques that use query decompositions and clever tricks to achieve strictly better space requirement and formal delay guarantees; *(ii)* our algorithms are applicable to arbitrary hypergraphs as compared to simple graph patterns over binary relations. Most closely related to our setting are [28] and [44]. Algorithm in [28] is fundamentally different from ours. It uses an adaptation of Lawler-Murty's procedure to generate candidate output tuples which is also a source of inefficiency given that it ignores query structure. [44] presented a novel anytime algorithm for enumerating *homomorphic tree patterns* with worst case delay and space guarantees where the ranking function is sum of weights of input tuples that contribute to an output tuple. Their algorithm also generates candidate output tuples with different scores and sorts them via a priority queue. However, the candidate generation phase is expensive and can be improved substantially, as we show in this paper. Our algorithm also generalizes the approach of prior work that showed how to find k shortest paths in a graph [19] and may be useful to other problems in DP [39] and DGM [11] where ranked enumeration is useful.

Rank aggregation algorithms. Top-k processing over ranked lists of objects has a rich history. The problem was first studied by Fagin et al. [20, 21] where the database consists of N objects and m ranked streams, each containing a ranking of the N objects with the goal of finding the top-k results for coordinate monotone functions. The authors proposed Fagin's algorithm (FA) and Threshold algorithm (TA), both of which were shown to be instance optimal for database access cost under sorted list access and random access model. This model would be applicable to our setting only if $Q(D)$ is already computed and materialized. More importantly, TA can only give $O(N)$ delay guarantee using $O(N)$ space. [32] extended the problem setting to the case where we want to enumerate top-k answers for t-path query. The first proposed algorithm J^* uses an iterative deepening mechanism that pushes the most promising candidates into a priority queue. Unfortunately, even though the algorithm is instance optimal with respect to number of sorted access over each list, the delay guarantee is $\Omega(|Q(D)|)$ with space requirement $S = \Omega(|Q(D)|)$. A second proposed algorithm J^*_{PA} allows random access over each sorted list. J^*_{PA} uses a dynamic threshold to decide when to use random access over other lists to find joining tuples versus sorted access but does not improve formal guarantees.

Query enumeration. The notion of constant delay query enumeration was introduced by Bagan, Durand and Grandjean in [3]. In this setting, preprocessing time is supposed to be much smaller than the time needed to evaluate the query (usually, linear in the size of the database), and the delay between two output tuples may depend on the query, but not on the database. This notion captures the *intrinsic hardness* of query structure. For an introduction to this topic and an overview of the state-of-the-art we refer the reader to the survey [40, 42]. Most of the results in existing works focus only on lexicographic enumeration of query results where the ordering of variables cannot be arbitrarily chosen. Transferring the static setting enumeration results to under updates has also been a subject of recent interest [7, 6].

Factorized databases. Following the landmark result of [36] which introduced the notion of using the logical structure of the query for efficient join evaluation, a long line of research has benefited from its application to learning problems and broader classes of queries [5, 4, 35, 16, 27, 14, 15]. The core idea of factorized databases is to convert an arbitrary query into an acyclic query by finding a query decomposition of small width. This width parameter controls the space and pre-processing time required in order to build indexes allowing for constant delay enumeration. We build on top of factorized representations and integrate ranking functions in the framework to enable enumeration beyond lexicographic orders.

7 Conclusion

In this paper, we study the problem of CQ result enumeration in ranked order. We combine the notion of query decompositions with certain desirable properties of ranking functions to enable logarithmic delay enumeration with small preprocessing time. The most natural open problem is to prove space lower bounds to see if our algorithms are optimal at least for certain classes of CQs. An intriguing question is to explore the full continuum of time-space tradeoffs. For instance, for any compatible ranking function with the 4-path query and $T_P = O(N)$, we can achieve $\delta = O(N^{3/2})$ with space $S_e = O(N)$ and $\delta = O(\log N)$ with space $S_e = O(N^2)$. The precise tradeoff between these two points and its generalization to arbitrary CQs is unknown. There also remain several open question regarding how the structure of ranking functions influences the efficiency of the algorithms. In particular, it would be interesting to find fine-grained classes of ranking functions which are more expressive than totally decomposable, but less expressive than coordinate decomposable. For instance, the ranking function $f(x, y) = |x - y|$ is not coordinate decomposable, but it is *piecewise* coordinate decomposable on either side of the global minimum critical point for each x valuation.

References

1 Mahmoud Abo Khamis, Hung Q Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 429–444. ACM, 2017.

2 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007.

4 Nurzhan Bakibayev, Tomáš Kočiskỳ, Dan Olteanu, and Jakub Závodnỳ. Aggregation and ordering in factorised databases. *Proceedings of the VLDB Endowment*, 6(14):1990–2001, 2013.

5 Nurzhan Bakibayev, Dan Olteanu, and Jakub Závodnỳ. Fdb: A query engine for factorised relational databases. *Proceedings of the VLDB Endowment*, 5(11):1232–1243, 2012.

6 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 303–318. ACM, 2017.

7 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering fo+ mod queries under updates on bounded degree databases. *ACM Transactions on Database Systems (TODS)*, 43(2):7, 2018.

8 Endre Boros, Benny Kimelfeld, Reinhard Pichler, and Nicole Schweikardt. Enumeration in Data Management (Dagstuhl Seminar 19211). *Dagstuhl Reports*, 9(5):89–109, 2019. doi:10.4230/DagRep.9.5.89.

9 David Bremner, Timothy M Chan, Erik D Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. Necklaces, convolutions, and x+ y. In *European Symposium on Algorithms*, pages 160–171. Springer, 2006.

10 Lijun Chang, Xuemin Lin, Wenjie Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. Optimal enumeration: Efficient top-k tree matching. *Proceedings of the VLDB Endowment*, 8(5):533–544, 2015.

11 Amrita Roy Chowdhury, Theodoros Rekatsinas, and Somesh Jha. Data-dependent differentially private parameter learning for directed graphical models. In *International Conference on Machine Learning*, pages 1939–1951. PMLR, 2020.

12 Sara Cohen and Yehoshua Sagiv. An incremental algorithm for computing ranked full disjunctions. *Journal of Computer and System Sciences*, 73(4):648–668, 2007.

13 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

14 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Fast join project query evaluation using matrix multiplication. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1213–1223, 2020.

15 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Enumeration algorithms for conjunctive queries with projection. In *To appear the the proceedings of ICDT '21 Proceedings*, 2021.

16 Shaleen Deep and Paraschos Koutris. Compressed representations of conjunctive query results. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 307–322. ACM, 2018.

17 Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. *arXiv preprint arXiv:1902.02698*, 2019.

18 Erik D Demaine and Joseph O'Rourke. Open problems from cccg 2005. In *Canadian Conference on Computational Geometry*, pages 75–80, 2005.

19 David Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673, 1998.

20 Ronald Fagin. Combining fuzzy information: an overview. *ACM SIGMOD Record*, 31(2):109–118, 2002.

21 Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

22 Michael L Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1(4):355–361, 1976.

23 Manish Gupta, Jing Gao, Xifeng Yan, Hasan Cam, and Jiawei Han. Top-k interesting subgraph discovery in information networks. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 820–831. IEEE, 2014.

24 John E Hopcroft, Jeffrey D Ullman, and AV Aho. The design and analysis of computer algorithms, 1975.

25 Ihab F Ilyas, George Beskales, and Mohamed A Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

26 Ihab F Ilyas, Rahul Shah, Walid G Aref, Jeffrey Scott Vitter, and Ahmed K Elmagarmid. Rank-aware query optimization. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 203–214. ACM, 2004.

27 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 375–392, 2020.

28 Benny Kimelfeld and Yehoshua Sagiv. Incrementally computing ordered answers of acyclic conjunctive queries. In *International Workshop on Next Generation Information Technologies and Systems*, pages 141–152. Springer, 2006.

29 Chengkai Li, Kevin Chen-Chuan Chang, Ihab F Ilyas, and Sumin Song. Ranksql: query algebra and optimization for relational top-k queries. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 131–142. ACM, 2005.

30 Chengkai Li, Mohamed A Soliman, Kevin Chen-Chuan Chang, and Ihab F Ilyas. Ranksql: supporting ranking queries in relational database management systems. In *Proceedings of the 31st international conference on Very large data bases*, pages 1342–1345. VLDB Endowment, 2005.

31 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM (JACM)*, 60(6):42, 2013.

32 Apostol Natsev, Yuan-Chi Chang, John R Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, volume 1, pages 281–290, 2001.

33 Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 37–48. ACM, 2012.

34 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Record*, 42(4):5–16, 2013. doi:10.1145/2590989.2590991.

35 Dan Olteanu and Maximilian Schleich. Factorized databases. *ACM SIGMOD Record*, 45(2):5–16, 2016.

36 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2, 2015. doi:10.1145/2656335.

37 Yan Qi, K Selçuk Candan, and Maria Luisa Sapino. Sum-max monotonic ranked joins for evaluating top-k twig queries on weighted data graphs. In *Proceedings of the 33rd international conference on Very large data bases*, pages 507–518. VLDB Endowment, 2007.

38 Christopher Re, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 886–895. IEEE, 2007.

39 Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. Crypt?: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 603–619, 2020.

40 Luc Segoufin. Enumerating with constant delay the answers to a query. In *Proceedings of the 16th International Conference on Database Theory*, pages 10–20. ACM, 2013.

41 Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Record*, 44(1):10–17, 2015. doi:10.1145/2783888.2783894.

42 Luc Segoufin. Constant delay enumeration for conjunctive queries. *ACM SIGMOD Record*, 44(1):10–17, 2015.

43 William L Steiger and Ileana Streinu. A pseudo-algorithmic separation of lines from pseudo-lines. *Inf. Process. Lett.*, 53(5):295–299, 1995.

44 Xiaofeng Yang, Deepak Ajwani, Wolfgang Gatterbauer, Patrick K Nicholson, Mirek Riedewald, and Alessandra Sala. Any-k: Anytime top-k tree pattern retrieval in labeled graphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 489–498. International World Wide Web Conferences Steering Committee, 2018.

45 Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Finding top-k maximal cliques in an uncertain graph, 2010.

Towards Optimal Dynamic Indexes for Approximate (and Exact) Triangle Counting

Shangqi Lu
The Chinese University of Hong Kong, China

Yufei Tao
The Chinese University of Hong Kong, China

Abstract

In ICDT'19, Kara, Ngo, Nikolic, Olteanu, and Zhang gave a structure which maintains the number T of triangles in an undirected graph $G = (V, E)$ along with the edge insertions/deletions in G. Using $O(m)$ space ($m = |E|$), their structure supports an update in $O(\sqrt{m} \log m)$ amortized time which is optimal (up to polylog factors) subject to the OMv-conjecture (Henzinger, Krinninger, Nanongkai, and Saranurak, STOC'15). Aiming to improve the update efficiency, we study:

- *the optimal tradeoff between update time and approximation quality.* We require a structure to provide the (ϵ, Γ)-*guarantee*: when queried, it should return an estimate t of T that has relative error at most ϵ if $T \geq \Gamma$, or an absolute error at most $\epsilon \cdot \Gamma$, otherwise. We prove that, under any $\epsilon \leq 0.49$ and subject to the OMv-conjecture, no structure can guarantee $O(m^{0.5-\delta}/\Gamma)$ expected amortized update time and $O(m^{2/3-\delta})$ query time simultaneously for any constant $\delta > 0$; this is true for $\Gamma = m^c$ of any constant c in $[0, 1/2)$. We match the lower bound with a structure that ensures $\tilde{O}((1/\epsilon)^3 \cdot \sqrt{m}/\Gamma)$ amortized update time with high probability, and $O(1)$ query time.
- *(for exact counting) how to achieve arboricity-sensitive update time.* For any $1 \leq \Gamma \leq \sqrt{m}$, we describe a structure of $O(\min\{\alpha m + m \log m, (m/\Gamma)^2\})$ space that maintains T precisely, and supports an update in $\tilde{O}(\min\{\alpha + \Gamma, \sqrt{m}\})$ amortized time, where α is the largest arboricity of G in history (and does not need to be known). Our structure reconstructs the aforementioned ICDT'19 result up to polylog factors by setting $\Gamma = \sqrt{m}$, but achieves $\tilde{O}(m^{0.5-\delta})$ update time as long as $\alpha = O(m^{0.5-\delta})$.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory)

Keywords and phrases Triangle Counting, Data Structures, Lower Bounds, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.6

Related Version *Full Version*: `http://www.cse.cuhk.edu.hk/~taoyf/paper/icdt21.pdf`

Funding This work was supported by a research grant under the title "Monitoring Patterns on Dynamic Graphs: Algorithms and Systems" from Alibaba Group.

1 Introduction

In the *dynamic approximate triangle counting* (DATC) problem, we want to maintain a data structure on an undirected graph $G = (V, E)$ to support

- **update**(e): either adds a new edge e or removes an existing edge e;
- **query**: returns an estimate t of the number T of *triangles* (i.e., 3-cliques) in G. Specifically, setting $m = |E|$, we require that the estimate t should satisfy an $(\epsilon, \Gamma(m))$-*guarantee*:

$$|t - T| \leq \begin{cases} \epsilon \cdot T & \text{if } T \geq \Gamma(m) \\ \epsilon \cdot \Gamma(m) & \text{otherwise} \end{cases} \quad (1)$$

  where ϵ is a parameter of the structure satisfying $0 < \epsilon \leq 1$, and $\Gamma(m)$ a non-descending function of m satisfying

- $\Gamma(m) \geq 1$
- $\Gamma(c \cdot m) = O(\Gamma(m))$ for any constant $c > 1$.

The query is allowed to fail with probability at most $1/m^2$.

Unless there is a need to emphasize on the parameter m, we will write function $\Gamma(m)$ simply as Γ. The (ϵ, Γ)-guarantee, phrased differently, requires that the estimate t should have a relative error at most ϵ or an absolute error at most $\epsilon \cdot \Gamma$.

The *dynamic exact triangle counting* (DETC) problem is defined analogously except that the value t returned by a query should always be equal to T.

Notations and math conventions. Throughout the paper, $\mathbb{N}$ is the set of integers, $[x]$ denotes the set $\{1, 2, ..., x\}$ for an integer $x \geq 1$, $\tilde{O}(.)$ suppresses a polylog m factor, $\{u, v\}$ represents an undirected edge between vertices u and v, while a directed edge from u to v is represented as (u, v). An event occurs *with high probability* (w.h.p.) if its probability is at least $1 - 1/m^2$.

1.1 Motivation

Triangle counting is equivalent to computing the output size of the conjunctive query

$$ans(a, b, c) \quad = \quad R_1(a, b), R_2(a, c), R_3(b, c). \tag{2}$$

DETC can be easily reduced to the above query by duplicating E three times. Conversely, query (2) can be reduced to DETC as follows. Suppose that relations R_1, R_2, R_3 have schemes $\{\texttt{A}, \texttt{B}\}$, $\{\texttt{A}, \texttt{C}\}$, and $\{\texttt{B}, \texttt{C}\}$, respectively, where attributes `A`, `B`, and `C` have disjoint domains. Create a graph $G = (V, E)$ such that (i) V contains a vertex for every distinct value of `A`, `B`, `C`, and (ii) E has an edge $\{u, v\}$ for every tuple (u, v) of R_1, R_2, R_3. It is easy to verify that each tuple (a, b, c) in the query result corresponds to a unique triangle in G, and vice versa. Inserting/deleting a tuple is translated to an edge update in G.

Our initial motivation stemmed from two recent results on DETC. Subject to the *OMv conjecture* (Section 1.2), Henzinger, Krinninger, Nanongkai, and Saranurak showed [20] (long version [21]) that no structure with $O(m^{0.5-\delta})$ amortized update time can guarantee $O(m^{1-\delta})$ query time, for any constant $\delta > 0$. Kara, Ngo, Nikolic, Olteanu, and Zhang [27] matched this lower bound with a linear-space structure of $O(\sqrt{m} \log m)$ amortized update time[1] and $O(1)$ query time.

$O(\sqrt{m} \log m)$ update time is rather expensive for practical applications. We thus ask:

> **Question 1:** How much loss of accuracy is necessary, if we want to (significantly) reduce the update cost of [27]?
> **Question 2:** If we insist on exact counting, how to derive an update bound using certain intrinsic parameters of G which can be $o(\sqrt{m})$ for many practical inputs?

1.2 Related Work

Upper Bounds. Kopelowitz et al. [28] studied the following *dynamic set intersection size* problem. Define $\mathcal{C}$ as a collection of non-empty sets $S_1, S_2, ..., S_\ell$ for some $\ell \geq 1$ (the domain of the elements therein is unimportant). Set $m = \sum_{S \in \mathcal{C}} |S|$. Given distinct $i, j \in [\ell]$, a *query*

[1] In [27], the amortized update complexity was stated as $O(\sqrt{m})$, assuming that dictionary search on a set of elements can be performed in constant time by a structure that can be updated also in constant time. Removing the assumption with hashing would degrade the update guarantee into an expected bound; doing so with a binary search tree would introduce a logarithmic factor.

reports the number of elements in $S_i \cap S_j$. We want to maintain a structure to support not only queries and also updates (element insertions/deletions) in the sets of $\mathcal{C}$. The structure of [28] uses $O(m)$ space, performs an update in $\tilde{O}(\sqrt{m})$ time, and answers a query in $\tilde{O}(\sqrt{m})$ time.[2] This structure can be deployed to perform DETC with the same guarantees as [27], up to polylog factors.

Eppstein and Spiro [17] described a DETC structure that supports a query in $O(1)$ time, and an update in $O(h \log m)$ time, where h is the *h-index* of G at the time of the update.[3] The update cost compares favorably with the structure of [27] (Section 1.1) because h is always $O(\sqrt{m})$ but can be far less than $\sqrt{m}$. However, the structure of [17] consumes $O(mh)$ space, while that of [27] needs only $O(m)$ space.

The DETC problem – equivalently, conjunctive query (2) – is a special form of the first-order queries studied by Berkholz et al. [8]. When applied to DETC, their structure performs an update in $\tilde{O}(1)$ time and a query in constant time, when the maximum degree d of the vertices is a constant. In general, however, the update time of [8] is $2^{d^{O(1)}}$ which is much higher than $\sqrt{m}$ even for moderate d. Note that the objective of [8] is to achieve results of this form over a broad class of queries on *sparse* databases (rather than just DETC).

In the *static* scenario where no updates are allowed, the fastest algorithm for exact triangle counting is still the classic $O(m^{2\omega/(\omega+1)})$-time algorithm of Alon, Yuster, and Zwick [1], where $\omega < 2.373$ is the exponent of matrix multiplication. Chiba and Nishizeki [13] described an algorithm of time $O(\alpha m)$ where α is the *arboricity* of G, which is the smallest number of edge-disjoint forests that cover all the edges in G; in general, α is between 1 and $\lceil\sqrt{m}\rceil$. For approximate counting up to relative error ϵ, Eden, Levi, Ron, and Seshadhri [16] gave an algorithm of $\tilde{O}((1/\epsilon)^2 \cdot m^{1.5}/T)$ time. This result can be generalized to counting arbitrary subgraphs; see the work of Assadi, Kapralov, and Khanna [2] and of Chen and Yi [12].

There is a line of research on approximate triangle counting with a *stream algorithm* that makes one or constant passes over E (see [3–5, 9, 11, 15, 18, 23–25, 31, 34, 35, 37] and the references therein). The main purpose there is to minimize the amount of space used. *One-pass* algorithms on *arbitrarily-ordered* streams (i.e., edges arriving in any order) can be used to deal with DATC when only insertions are present. However, in that scenario, Braverman, Ostrovsky, and Vilenchik [9] showed that $\Omega(m)$ space is compulsory even to distinguish between $T = 0$ and $T = \Omega(|V|)$. This implies the necessity of retaining E entirely in the worst case. Our DATC problem complements [9] by asking: as E must be stored anyway, how to organize it properly to permit fast updates?

There have been works on approximate triangle counting on a *dynamic* stream (arbitrary edges insertions and deletions). Bulteau, Froese, Kutzkov, and Pagh [10] developed a structure of $\tilde{O}((1/\epsilon)^2 \cdot \sqrt{m} \cdot P_2/T)$ space that has constant query time but $\tilde{O}((1/\epsilon)^2 \cdot P_2/T)$ update time, where P_2 is the number of 2-paths in G. Another structure due to Manjunath, Mehlhorn, Panagiotou, and Sun [30] uses $\tilde{O}(\text{poly}(1/\epsilon) \cdot m^3/T^2)$ space, and achieves constant query time and $\tilde{O}(\text{poly}(1/\epsilon) \cdot m^3/T^2)$ update time (see also [26]). These structures are applicable to DATC, but their update time is quite large compared to our results (Section 1.3). It should be noted, however, that the focus of [10, 26, 30] is to understand when the space can be made $o(m)$, rather than the update-query tradeoff.

[2] Precisely speaking, Kopelowitz et al. [28] considered a different type of queries, which return *whether* $S_i \cap S_j$ is empty (as opposed to $|S_i \cap S_j|$). However, their structure can be easily adapted to achieve the stated guarantees on the dynamic set intersection size problem.

[3] The h-index is the maximum integer x such that G has x vertices of degree at least x.

A natural attempt to perform DATC on $G = (V, E)$ is to take a random subset $E' \subseteq E$, build an *exact* counting structure to monitor the number T' of triangles in $G' = (V, E')$, and then scale T' up appropriately to estimate the number of triangles in G. To our knowledge, the most promising approach in this direction is the *colorful triangle sampling* technique by Pagh and Tsourakakis [32], originally proposed for parallel computation. In our contexts, the technique is applicable if Γ is sufficiently large. This can be best illustrated by fixing ϵ to a constant; when $\Gamma \geq c|V| \log_2 |V|$ for some constant c, the technique (combined with [27]) gives a structure supporting a query in constant time and an update in $\tilde{O}(\sqrt{m} \cdot \max\{\frac{|V|^{1.5}}{\Gamma^{1.5}}, \frac{1}{\Gamma^{0.75}}\})$ time w.h.p. This bound will be strictly improved by our methods.

Lower bounds. In the *online boolean matrix-vector multiplication* (OMv) problem, an algorithm first spends poly(n) time preprocessesing an $n \times n$ boolean matrix $\boldsymbol{M}$, and is then required to compute $\boldsymbol{M}\boldsymbol{v}_i$ ($i \in [n]$) where each $\boldsymbol{v}_i$ is an $n \times 1$ boolean vector.[4] Vector $\boldsymbol{v}_{i+1}$ ($i \geq 1$) is revealed only after the algorithm has output $\boldsymbol{M}\boldsymbol{v}_i$. The *cost* is the total time spent on the n vectors.

> **OMv-conjecture [21]:** no algorithm can solve the problem with probability at least 2/3 using *subcubic* cost $O(n^{3-\delta})$ for any constant $\delta > 0$.

The conjecture explains in a remarkable manner the computational hardness of a great variety of problems [21], and gives rise to the tight (conditional) lower bound on DETC mentioned in Section 1.1 (see [7] for the conjecture's implications on conjunctive queries when the update time has to be $\tilde{O}(1)$).

It has been shown [21] that the OMv conjecture implies another well-known conjecture formulated by Patrascu [33] on the *multiphase problem* (namely, if the former is correct, so is the latter, which means that the former is at least as hard to prove as the latter). Patrascu's conjecture has been utilized to establish (conditional) lower bounds on *dynamic set intersection emptiness* [19, 28, 29], which can be converted to lower bounds on DETC, but they are not tight (we will elaborate on this in Section 2). Indeed, many of the lower bounds obtained from Patrascu's conjecture can be strengthened with OMv (see [21] for a comprehensive list); the same phenomenon also applies to the DATC lower bound (Theorem 1) developed in this paper (more details in Section 2).

1.3 Our Results

DATC. Regarding Question 1 (Section 1.1), we first prove a conditional lower bound:

▶ **Theorem 1.** *Consider the DATC problem where $\epsilon \leq 0.49$ and $\Gamma = m^c$ for an arbitrary constant c satisfying $0 \leq c < 1/2$. Subject to the OMv-conjecture, no DATC structure can ensure $O(m^{0.5-\delta}/\Gamma)$ amortized update time and $O(m^{\frac{2}{3}-\delta})$ query time simultaneously, where $\delta > 0$ is an arbitrary constant. This is true even if the amortized update time holds only in expectation.*

We are able to match the lower bound with:

▶ **Theorem 2.** *There is a DATC structure that ensures $\tilde{O}((1/\epsilon)^3 \cdot \sqrt{m}/\Gamma)$ amortized update time w.h.p. and $O(1)$ query time. The space of the structure is $\tilde{O}(m + (1/\epsilon)^2 \cdot m^{1.5}/\Gamma)$.*

[4] Additions and multiplications are as in the boolean semi-ring.

For constant $\epsilon \leq 0.49$, Theorems 1 and 2 together give the full tradeoff between update time and the approximation quality (subject to the OMv-conjecture). As a pleasant implication, for constant ϵ Theorem 2 shows that one can achieve $\tilde{O}(1)$ amortized update time and $O(1)$ query time by setting $\Gamma = \sqrt{m}$; in other words, we never have to worry about $\Gamma > \sqrt{m}$ (simply lower such Γ to $\sqrt{m}$). It is interesting to note, in retrospect, that the constant c in Theorem 1 does not reach 1/2.

DETC. We address Question 2 by giving a new structure whose performance depends on the *arboricity* of G (Section 1.2):

▶ **Theorem 3.** *For any monotonic function $\Gamma(m)$ satisfying $1 \leq \Gamma(m) \leq \sqrt{m}$ and $\Gamma(c \cdot m) = O(\Gamma(m))$, there is a DETC structure of $O(\min\{\alpha m + m \log m, (\frac{m}{\Gamma(m)})^2\})$ space that supports an update in $\tilde{O}(\min\{\alpha + \Gamma(m), \sqrt{m}\})$ amortized time, and a query in $O(1)$ time, where α is the largest arboricity of G in history. This holds even if α is unknown.*

By setting $\Gamma = \sqrt{m}$, we reconstruct the result of [27] up to polylog factors; on the other hand, we can do significantly better when α is small, i.e., G is sparse. In particular, when G is a planar graph, $\alpha = O(1)$; thus our structure achieves $O(m \log m)$ space, $\tilde{O}(1)$ amortized update time, and constant query time. The arboricity of a graph is always bounded by the h-index, but can be considerably lower, e.g., a planar graph can have an h-index of $\Theta(\sqrt{m})$; our structure is, therefore, not subsumed by [17] (Section 1.2). Similarly, even a planar graph can have a maximum vertex degree of $\Theta(|V|)$; our result is, therefore, not subsumed by [8] either. Interestingly, if α is known in advance, by setting $\Gamma = \alpha$, we obtain a structure occupying $\tilde{O}(\min\{\alpha m, m^2/\alpha^2\}) = \tilde{O}(m^{4/3})$ space that supports an update in $\tilde{O}(\alpha)$ time and ensures constant query time.

2 Hardness of Dynamic Approximate Triangle Counting

In this section, we will prove:

▶ **Lemma 4.** *Consider the DATC problem with $\epsilon = 0.49$ and $\Gamma = m^c$ for an arbitrary constant c satisfying $0 \leq c < 1/2$. Subject to the OMv-conjecture, no structure can guarantee $O(m^{0.5-\delta-c})$ expected amortized update time and $O(m^{1-2c/3-\delta})$ query time, where $\delta > 0$ can be an arbitrarily small constant.*

Theorem 1 is a corollary of Lemma 4, noticing that (i) $1 - 2c/3 > 2/3$ for $c < 1/2$, and (ii) any solution that works for $\epsilon < 0.49$ must also work for $\epsilon = 0.49$. To prove the lemma, we will consider the *dynamic triangle detection* (DTD) problem, where we want to store G in a data structure to support:

- **update**(e): either adds a new edge e or removes an existing edge e;
- **query**: returns a single bit indicating whether G has any triangles at all. The query is allowed to fail with probability at most $1/m^2$.

The lemma below was first established in [21]:

▶ **Lemma 5** ([21]). *Subject to the OMv-conjecture, no DTD structure can guarantee $O(m^{0.5-\delta})$ amortized update time and $O(m^{1-\delta})$ query time, where $\delta > 0$ can be an arbitrarily small constant. This is true even if the amortized update time holds only in expectation.*[5]

[5] The statement in [21] (see Corollary 3.4 therein) does not contain the second sentence. Furthermore, the DTD query in [21] is not allowed to fail. However, it is easy to extend their argument to prove Lemma 5. We provide a complete proof in the full version of this paper, which can be found on the homepage of the second author.

Suppose that algorithm $\mathcal{A}$ is able to maintain a DATC structure – on our instance where $\epsilon = 0.49$ and $\Gamma = m^c$ – which supports an update in $O(m^{0.5-\delta'}/\Gamma) = O(m^{0.5-\delta'-c})$ expected amortized time and a query in $O(m^{1-2c/3-\delta'})$ time for some $\delta' > 0$. We will deploy $\mathcal{A}$ to obtain a DTD structure that contradicts Lemma 5.

Establishing Lemma 4. Henceforth, denote by G the input graph to the DTD problem, and by m the number of edges in G. Given an integer parameter $x \geq 1$, we define an *image graph* [15] G' as follows:

- for each vertex u in G, create x *image vertices* in G';
- for each edge $\{u, v\}$ in G, create x^2 *image edges* in G' by connecting every image vertex of u and every image vertex of v.

The total number of edges in G' equals $m' = x^2 m$. Observe that if G has T triangles, then the number of triangles in G' is $T' = x^3 T$.

We now proceed to explain how to support updates and DTD queries on G. For this purpose, let us first assume that $M \leq m \leq 2M$ for some integer $M \geq 1$. The assumption will be removed with global rebuilding, as explained later.

We choose:

$$x = (2M)^{\frac{c}{3-2c}}. \tag{3}$$

with which $m' = x^2 m = \Theta(m^{\frac{3}{3-2c}})$.

We apply $\mathcal{A}$ to build a DATC structure on G' (with $\epsilon = 0.49$ and $\Gamma = m'^c$). Given an **update**(e) on G, we use $\mathcal{A}$ to insert/delete all the x^2 image edges of e in G' in expected amortized time

$$O(m'^{0.5-\delta'-c} \cdot x^2) = O(m^{\frac{2c}{3-2c} + \frac{3}{3-2c}(\frac{1}{2}-\delta'-c)}) = O(m^{\frac{1}{2}-\frac{3\delta'}{3-2c}}).$$

To explain how to answer a DTD query, we will need:

▶ **Proposition 6.** $\epsilon m'^c < x^3/2$.

Proof. First note that $m' = x^2 m \leq (2M)^{\frac{2c}{3-2c}} \cdot (2M) = (2M)^{\frac{3}{3-2c}}$. Hence, $\epsilon m'^c$ is at most $0.49 \cdot (2M)^{\frac{3c}{3-2c}} < x^3/2$. ◀

G has a triangle if and only if G' has at least $T' \geq x^3$ triangles. Given a DTD query on G, we run $\mathcal{A}$ to detect whether $T' \geq x^3$. For this purpose, it suffices to issue a DATC query on G'. The output t of the DATC query is greater than $x^3/2$ if and only if $T' \geq x^3$. This is because

- when $T' < x^3$, it must hold that $T' = 0$, in which case t can be at most $\epsilon \cdot \Gamma(m') = \epsilon m'^c < x^3/2$ (Proposition 6);
- when $T' \geq x^3$, $t \geq (1-\epsilon)T' \geq (1-\epsilon)x^3 > x^3/2$.

By our assumptions on $\mathcal{A}$, the DATC query runs in time

$$O(m'^{1-\frac{2c}{3}-\delta'}) = O(m^{\frac{3}{3-2c}(1-\frac{2c}{3}-\delta')}) = O(m^{1-\frac{3\delta'}{3-2c}}).$$

It remains to remove the assumption $M \leq m \leq 2M$. For this purpose, it suffices to destroy and rebuild the DATC structure whenever m reaches M or $2M$. The value of M for the new structure is set to $2m/3$. This makes sure $\Omega(M)$ updates on G must have happened before the next reconstruction. Standard amortization arguments show that the amortized update time is still $O(m^{\frac{1}{2}-\frac{3\delta'}{3-2c}})$ in expectation.

We thus have obtained a DTD structure with expected amortized update time $O(m^{0.5-\delta})$ and query time $O(m^{1-\delta})$ with $\delta = \frac{3\delta'}{3-2c}$, contradicting Lemma 5. This completes the proof of Lemma 4.

Remarks. A weaker lower bound would result from Patrascu's multiphase conjecture [33]. Consider, for simplicity, $c = 0$ (essentially, exact counting) in which case the strongest lower bound derived with that conjecture [28,29] asserts that no structure can guarantee $O(m^{1/3-\delta})$ update and query time simultaneously[6]. This is also the best we can prove by executing our argument on the multiphase conjecture, but is worse than Theorem 1 by a polynomial factor. Finally, it is worth mentioning that our argument actually works for any $\epsilon < 0.5$.

3 A Structure for Dynamic Approximate Triangle Counting

This section presents a DATC structure which achieves the performance in Theorem 2.

3.1 Overview

We will start by describing a "folklore" algorithm (see Section 3.6 for a discussion) for approximate triangle counting on a *static* graph $G = (V, E)$. Denote by $d(u)$ the degree of vertex $u \in V$. Define an ordering $\prec$ on V: $u \prec v$ if $d(u) < d(v)$, breaking ties by id. Orient G by pointing each edge $\{u, v\} \in E$ from u to v where $u \prec v$. Let E^+ be the set of directed edges thus obtained, and define $G^+ = (V, E^+)$ as the resulting directed graph. Denote by $d^+(u)$ the out-degree of $u \in V$ in G^+; it must hold that $d^+(u) = O(\sqrt{m})$.

To estimate the number T of triangles, initialize $\Lambda = 0$, and repeat the following $s = \tilde{O}((1/\epsilon)^2 \cdot m^{1.5}/T)$ times:

1. Take an edge $(u, v) \in E^+$ and then an out-neighbor w of u, both uniformly at random (note that v may be w). We will refer to (u, v, w) as a *random tuple*.
2. Add the *contribution* of (u, v, w) to Λ, which is $d^+(u)$ if $(v, w) \in E^+$, or 0 otherwise.

Finally, return $\Lambda \cdot (m/s)$ as the estimate, guaranteed to enjoy a relative error at most ϵ w.h.p. Our structure dynamizes the above algorithm, as outlined next.

Standard ideas. We can replace T with Γ (Section 1), and maintain a set S of $s = \tilde{O}((1/\epsilon)^2 \cdot m^{1.5}/\Gamma)$ random tuples, as well as the sum Λ of their contributions. Inserting/deleting an edge $\{u, v\}$ may flip the directions of many edges, rendering it expensive to keep G^+ up-to-date. But the issue can be easily remedied: it suffices to flip an edge only after $\Omega(\min\{d(u), d(v)\})$ updates. For this purpose, we introduce a function D such that $D(u)$ approximates $d(u)$ up to a small constant factor for every $u \in V$. Accordingly, $\prec$ is redefined with respect to D: $u \prec v$ if $D(u) < D(v)$, breaking ties by id. We can then afford to materialize G^+ explicitly by updating it only when D changes.

$D(u)$ is adjusted when it ceases to approximate $d(u)$. When this happens, some edges of u in G^+ have their directions flipped, e.g., (u, v) becomes (v, u). A major challenge now enters the picture: *the altering of $d^+(v)$ may affect all the contributions of the random tuples (x, y, z) with $x = v$!* Specifically, each $(v, y, z) \in S$ may have already registered in Λ a contribution $d^+(v)$, which therefore must be modified. Unfortunately, we cannot afford to do so for all neighbors v of u.

New ideas. We overcome the above challenge by introducing another function D^+ such that $D^+(u)$ approximates $d^+(u)$ up to some small factor for every $u \in V$. For each random tuple $(u, v, w) \in S$, its contribution is either $D^+(u)$ – as opposed to $d^+(u)$ – or 0. Only

[6] A DETC structure with $O(m^{1/3-\delta})$ update and query time will lead to $t_i = O(N^{1/3-\delta})$ and $t_q = O(N^{1/3-\delta})$ in the context of Theorem 9 of [28], causing a contradiction there.

when $D^+(u)$ ceases to approximate $d^+(u)$ will we adjust the tuple's contribution in Λ. This "two-level approximation" (i.e., D and D^+) is the key in our solution to DATC. We will argue that D, D^+, S, and Λ can be maintained efficiently along with the edge updates.

3.2 Structure

Our discussion will assume that the number m of edges in G satisfies $M \leq m \leq 2M$ for some integer $M \geq 1$. The assumption can be removed by reconstructing our structure periodically.

Main structure. Let $D : V \to \mathbb{N}$ be a function such that for every $u \in V$:

$$D(u) \begin{cases} = 2 & \text{if } d(u) \leq 1 \\ \in [\frac{1}{2}d(u), \frac{3}{2}d(u)] & \text{otherwise.} \end{cases} \tag{4}$$

As mentioned, for two distinct vertices $u, v \in V$, $u \prec v$ if $D(u) < D(v)$, breaking ties by id. This gives rise to the directed graph $G^+ = (V, E^+)$ as defined in Section 3.1. Let $D^+ : V \to \mathbb{N}$ be another function such that for every $u \in V$:

$$D^+(u) \quad \in \quad \Big[(1 - \epsilon/2) \cdot d^+(u), (1 + \epsilon/2) \cdot d^+(u)\Big]. \tag{5}$$

During an edge insertion/deletion, function D (or D^+, resp.) may temporarily violate (4) (or (5), resp.), in which case we say that the function is *bad*. D (or D^+, resp.) is *good* when no violation occurs. At the beginning or right after reconstruction, $D^+(u) = d^+(u)$ for all $u \in V$; and $D(u) = d(u)$ if $d(u) \geq 2$, or 2 otherwise.

Set $s = \tilde{O}((1/\epsilon)^2 \cdot M^{1.5}/\Gamma(M))$; note that the function $\Gamma(.)$ is parameterized for the smallest possible $m = M$. Define S to be a set of s independent random tuples drawn from G^+ (Section 3.1). Each tuple $(x, y, z) \in S$ makes a *contribution*

$$f(x, y, z) \quad = \quad \begin{cases} D^+(x) & \text{if } (y, z) \in E^+ \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Set

$$\Lambda \quad = \quad \sum_{(x,y,z) \in S} f(x, y, z). \tag{7}$$

Given vertices $u, v \in V$, define:

$$\Xi_{u,v} \quad = \quad \sum_{(x,u,v) \in S} D^+(x) \tag{8}$$

where the summation is over the random tuples (x, y, z) satisfying $y = u, z = v$. The pair (u, v) is *active* if at least one such random tuple exists.

Our structure can be summarized as: (i) graphs G and G^+, (ii) functions D and D^+, (iii) the set S of random tuples, and (iv) the value of Λ, and values of $\Xi_{u,v}$'s for all active (u, v). It is worth pointing out that Λ and the $\Xi_{u,v}$'s do *not* imply the need to maintain the contribution function f in (6).

Filtered subsets of S. We will use "$\perp$" to denote a wildcard, and define the boolean expression "$u = \perp$" to be true for any $u \in V$. Given q_1, q_2, and q_3 where each q_i $(1 \leq i \leq 3)$ is either a vertex or a wildcard, we introduce:

$$S_{q_1,q_2,q_3} \quad = \quad \{(x, y, z) \in S \mid x = q_1, y = q_2, z = q_3\}$$

namely, the subset obtained by filtering S using q_1, q_2, q_3.

▶ **Lemma 7.** *All the statements below are true:*

- *For any $u \in V$, $|S_{u,\perp,\perp}| = \tilde{O}(d^+(u) \cdot s/m)$ w.h.p.*
- *For any $u, v \in V$ such that $(u, v) \in E^+$, $|S_{u,v,\perp}| = \tilde{O}(s/m)$ w.h.p.*
- *For any $u, v \in V$ such that $(u, v) \in E^+$, $|S_{u,\perp,v}| = \tilde{O}(s/m)$ w.h.p.*

Proof. A random tuple (x, y, z) satisfies $x = u$ if and only if (x, y) is an out-edge of u in G^+. As (x, y) is a random edge in G^+, it is an out-edge of u with probability $d^+(u)/m$. Due to independence, $|S_{u,\perp,\perp}|$ is $\tilde{O}(s \cdot d^+(u)/m)$ w.h.p., as stated in the first bullet.

To prove the 2nd (or 3rd, resp.) bullet, it suffices to show that (x, y, z) belongs to $|S_{u,v,\perp}|$ (or $|S_{u,\perp,v}|$, resp.) with probability $1/m$. This is obvious for $S_{u,v,\perp}$. For (x, y, z) to appear in $S_{u,\perp,v}$:

- (x, y) must be an out-edge of u, which happens with probability $d^+(u)/m$;
- z chooses v, which happens with probability $1/d^+(u)$.

Therefore, $\mathbf{Pr}[(x, y, z) \in S_{u,\perp,v}] = 1/m$. ◀

Auxiliary structures. We assume the availability of *auxiliary structures* for:

- Given any q_1, q_2 and q_3, retrieve the size of S_{q_1,q_2,q_3} in $\tilde{O}(1)$ time.
- Given any q_1, q_2, q_3 and an integer k between 1 and $|S_{q_1,q_2,q_3}|$, uniformly sample k tuples *without replacement* (WoR) from S_{q_1,q_2,q_3} in $\tilde{O}(k)$ time. By setting $k = |S_{q_1,q_2,q_3}|$, we can use the operation to extract the entire S_{q_1,q_2,q_3}.
- Given any $u, v \in V$, in $\tilde{O}(1)$ time either retrieve $\Xi_{u,v}$ or assert that (u, v) is not active.
- Generate a random tuple from G^+ in $\tilde{O}(1)$ time.

All the auxiliary structures can be implemented as simple variants of binary search trees (see Chapter 14 of [14]).

Space. The overall space consumption is clearly $O(m + s) = \tilde{O}(m + (1/\epsilon)^2 \cdot m^{1.5}/\Gamma(m))$, using the fact that $\Gamma(m) \leq \Gamma(2M) = O(\Gamma(M))$.

Query. We will prove in Appendix B:

▶ **Lemma 8.** *With probability at least $1 - 1/m^3$, the value $\Lambda \cdot (M/s)$ is an estimate satisfying the $(\epsilon, \Gamma(m))$ guarantee.*

A query can therefore be answered in constant time.

Remarks. The following subsections will explain how to support insertions. The deletion algorithm is similar, with details duly presented in Appendix C.

Our discussion will ignore the auxiliary structures because they are rudimentary; and their maintenance cost can be higher than that of S and $\{\Xi_{u,v} \mid \text{active } (u, v)\}$ by at most a logarithmic factor. Furthermore, when a tuple (x, y, z) is inserted/deleted in S, Λ and $\Xi_{y,z}$ can be updated accordingly in logarithmic time. We will, therefore, not discuss explicitly the modifications to Λ and $\{\Xi_{u,v} \mid \text{active } (u, v)\}$ caused by insertions/deletions in S.

3.3 Insertion: When D Will Still Be Good

Suppose that we are inserting an edge $\{u^*, v^*\}$ in G. After the insertion, $d(u^*)$ and $d(v^*)$ both increase by 1. In this section, we consider that *D is still good for the new $d(u^*)$ and $d(v^*)$*. Consequently, every existing edge in G^+ retains its direction. Without loss of generality, assume that $\{u^*, v^*\}$ points from u^* to v^* in G^+.

Rationale. How would this affect a random tuple $(x, y, z) \in S$? Recall that (x, y) is supposed to be drawn uniformly at random from E^+. Now that m has increased by 1, (x, y) should be replaced by (u^*, v^*) with probability $1/m$ (reservoir sampling [38]). If the replacement occurs, (x, y, z) is said to be *edge-replaced*; in this case, we take a (uniformly) random out-neighbor w of u^*, delete (x, y, z) from S, and add (u^*, v^*, w).

For a tuple (x, y, z) that is *not* edge-replaced, further processing is necessary in two cases:

- Case 1: $x = u^*$. Since u^* has got a new out-neighbor v^*, z (which is supposedly a random out-neighbor of x) should be replaced by v^* with probability $1/d^+(u^*)$. If the replacement happens, (x, y, z) is said to be *outneighbor-replaced*; in this case, we delete (x, y, z) from S and add (u^*, y, v^*) instead.
- Case 2: $y = u^*, z = v^*$. The new edge (u^*, v^*) completes the triangle formed by x, u^*, v^*. We should therefore increase Λ (see (7)) by $f(x, y, z) = D^+(x)$.

Algorithm 1 Pseudocode of the insertion algorithm.

algorithm `insert` (u^*, v^*) /* a new edge (u^*, v^*) has just been added to G^+ */
1. generate an integer k_1 following the binomial distribution $B(|S|, 1/m)$
2. $S_1 \leftarrow$ a size-k_1 WoR sample set of S; remove S_1 from S
3. generate an integer k_2 following the binomial distribution $B(|S_{u^*,\perp,\perp}|, 1/d^+(u^*))$
4. $S_2 \leftarrow$ a size-k_2 WoR sample set of $S_{u^*,\perp,\perp}$; remove S_2 from S
   /* the removal of each $(x, y, z) \in S_1 \cup S_2$ requires updating Λ and $\Xi_{y,z}$ */
5. increase Λ by Ξ_{u^*,v^*}
6. **repeat** k_1 **times**
7. add (u^*, v^*, w) into S where w is a (uniformly) random out-neighbor of u^*
   /* requires updating Λ and $\Xi_{v^*,w}$ */
8. **for** each $(u^*, y, z) \in S_2$ **do**
9. add (u^*, y, v^*) to S /* requires updating Λ and Ξ_{y,v^*} */

Insertion algorithm. Algorithm 1 presents the algorithm in pesudocode. To find the edge-replaced tuples, we cannot afford to toss a coin for each tuple in S. However, we do not have to; because the tuples in S are independent, it suffices *generate* how many – say k_1 – edge-replaced tuples there should be, and draw a WoR sample set S_1 of size k_1 from S. Here, k_1 follows the binomial distribution $B(|S|, 1/m)$, and can be generated in $\tilde{O}(1)$ time (see, e.g., [38]). Using the auxiliary structures, we can extract S_1 and remove the tuples therein from S (Lines 1-2) in $\tilde{O}(k_1)$ time where $k_1 = \tilde{O}(|S|/m) = \tilde{O}(s/m)$ w.h.p. The same idea also applies to outneighbor-replaced tuples in Case 1. The number k_2 of such tuples follows the binomial distribution $B(|S_{u^*,\perp,\perp}|, \frac{1}{d^+(u^*)})$; hence, $k_2 = \tilde{O}(|S_{u^*,\perp,\perp}|/d^+(u^*)) = \tilde{O}(s/m)$ w.h.p. (Lemma 7). From $S_{u^*,\perp,\perp}$, we extract a WoR sample set S_2 of size k_2 in $\tilde{O}(k_2) = \tilde{O}(s/m)$ time using the auxiliary structures; S_2 can be regarded as the set of outneighbor-replaced tuples, which are then removed from S in $\tilde{O}(s/m)$ time (Line 3-4). Increasing the value of Λ due to Case 2 can be accomplished by simply adding Ξ_{u^*,v^*} (defined in (8)) to Λ (Line 5). The value of Ξ_{u^*,v^*} can be retrieved in $\tilde{O}(1)$ time from the auxiliary structures. Lines 6-9 then replenish S for the random tuples in $S_1 \cup S_2$ removed earlier.

After the insertion, the out-degree $d^+(u^*)$ of u^* increases by 1. If $D^+(u^*)$ still satisfies (5), the insertion is complete. Otherwise, we call `fix-Dplus`(u^*) (introduced below) and finish. In summary, the insertion runs in $\tilde{O}(s/m)$ time, plus the cost of `fix-Dplus`(u^*).

Algorithm `fix-Dplus`(u). This algorithm has the following constraint:

Invariant: when called, $D^+(u)$ violates (5).

`fix-Dplus`(u) first makes a copy of the current $D^+(u)$ – denote the copy as $\mathcal{D}^+_{old}$ – and then resets $D^+(u)$ to $d^+(u)$. Accordingly, for every $(x, y, z) \in S$ with $x = u$, its contribution $f(x, y, z)$ may change from $\mathcal{D}^+_{old}$ to $d^+(u)$. This may affect Λ and every $\Xi_{v,w}$ where v and w are out-neighbors of u in G^+. To remedy all these, we first retrieve $S_{u,\perp,\perp}$, and then for every $(u, y, z) \in S_{u,\perp,\perp}$:

- if $(y, z) \in E^+$, increase Λ by $d^+(u) - \mathcal{D}^+_{old}$;
- increase $\Xi_{y,z}$ by $d^+(u) - \mathcal{D}^+_{old}$.

By Lemma 7, $S_{u,\perp,\perp} = \tilde{O}(d^+(u) \cdot s/m)$ w.h.p. This implies:

▶ **Lemma 9.** *The cost of* `fix-Dplus`(u) *is* $\tilde{O}(|\mathcal{D}^+_{old} - d^+(u)| \cdot s/(\epsilon m))$ *w.h.p.*

Proof. The cost of `fix-Dplus`(u) is $\tilde{O}(d^+(u) \cdot s/m)$. Next, we show $d^+(u) = O(|\mathcal{D}^+_{old} - d^+(u)|/\epsilon)$. Consider the two possibilities of how $D^+(u)$ can violate (5). If $\mathcal{D}^+_{old} > (1 + \epsilon/2) \cdot d^+(u)$, then $d^+(u) < (\mathcal{D}^+_{old} - d^+(u)) \cdot (2/\epsilon)$. On the other hand, if $\mathcal{D}^+_{old} < (1 - \epsilon/2) \cdot d^+(u)$, we have $d^+(u) < (d^+(u) - \mathcal{D}^+_{old}) \cdot (2/\epsilon)$. ◀

3.4 Insertion: When D Will Go Bad

Again, denote by $\{u^*, v^*\}$ the edge to be inserted. This time, we consider that D will be bad after $d(u^*)$ and $d(v^*)$ increase by 1. In other words, D will cease to satisfy (4) with respect to u^*, v^*, or both. Our strategy is *not* to perform the insertion immediately. Instead, we will first modify D to make sure that D will *still* be good after the insertion. Then, the insertion can be processed by the algorithm in Section 3.3.

Next, we will introduce an algorithm named `fix-D` which takes a vertex u as the parameter, and has the following constraint:

Invariant: when called:

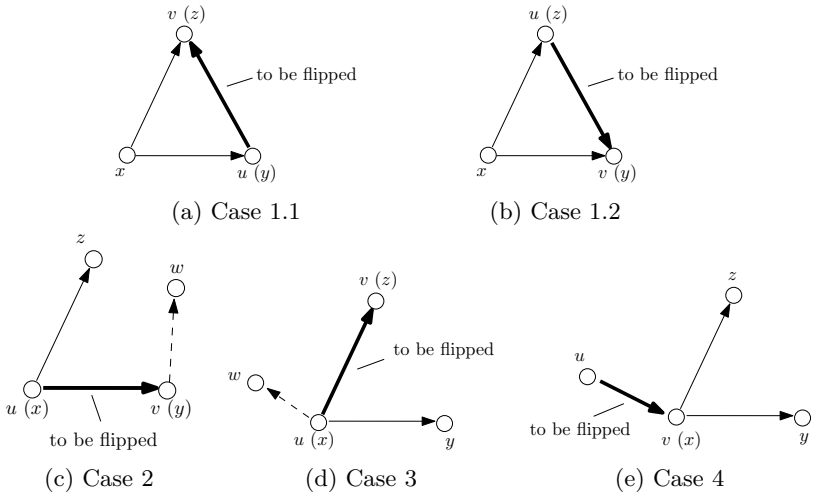
- D is good
- $D(u) < d(u)$ and $d(u) = O(D(u))$, and
- $d(u) - D(u) = \Omega(D(u))$.

At the end of `fix-D`(u), $D(u) = d(u)$, which ensures that $D(u)$ will still satisfy (4) even after $d(u)$ grows by 1. Thus, for the aforementioned insertion, we can simply invoke `fix-D`(u^*) and/or `fix-D`(v^*), depending on which will cause D to go bad.

Rationale behind `fix-D`(u). We increase $D(u)$ to $d(u)$. Recall that, for each neighbor v of u in G, the edge $\{u, v\}$ is given a direction in G^+. The increase of $D(u)$ may affect the direction: if the direction was (u, v) before, it may now be flipped to (v, u); on the other hand, if the direction was (v, u), it remains the same.

The direction flipping can invalidate S because a tuple in S may stop being a *random* tuple, or its contribution as in (6) may change (which will further affect Λ). To explain, fix a tuple $(x, y, z) \in S$, and suppose that an edge (u, v) is to be flipped to (v, u). Next, we enumerate all possible cases where modifications are necessary:

- Case 1: $x \neq u$ and $x \neq v$. (x, y, z) will remain as a random tuple. However, its contribution $f(x, y, z)$ is affected in two subcases:
  - Case 1.1: $y = u$ and $z = v$. $f(x, y, z)$ will drop from $D^+(x)$ to 0. Accordingly, Λ needs to be decreased by $D^+(x)$. See Figure 1(a).
  - Case 1.2: $y = v$ and $z = u$. $f(x, y, z)$ will grow from 0 to $D^+(x)$. Accordingly, Λ needs to be increased by $D^+(x)$. See Figure 1(b).



(a) Case 1.1 (b) Case 1.2 (c) Case 2 (d) Case 3 (e) Case 4

Figure 1 Different cases of `fix-D`.

- Case 2: $x = u$ and $y = v$. (x, y, z) will become invalid due to the disappearance of (x, y). The tuple (u, v, z) should be replaced by (v, u, w) where w is a (uniformly) random out-neighbor of v. See Figure 1(c).
- Case 3: $x = u$, $y \neq v$, and $z = v$. (x, y, z) will become invalid due to the disappearance of (x, z). The tuple (u, y, v) should be replaced by (u, y, w) where w is a (uniformly) random out-neighbor of u. See Figure 1(d).
- Case 4: $x = v$ (which implies $y \neq u$ and $z \neq u$). Since v has gained a new out-neighbor u, (x, y, z) may no longer be random. To remedy this, z should be replaced by u with probability $1/d^+(v)$. If the replacement occurs, the tuple (v, y, z) is said to be *outneighbor-replaced*. See Figure 1(e).

Algorithm `fix-D`(u). We start by setting $D(u) = d(u)$, flipping the edges of u in G^+ wherever needed.

Given each neighbor v of u in G such that $\{u, v\}$ was flipped, we

- (for Case 1) retrieve $\Xi_{u,v}$ and $\Xi_{v,u}$ (from the auxiliary structures), and increase Λ by $\Xi_{v,u} - \Xi_{u,v}$.
- (for Case 2) retrieve $S_{u,v,\perp}$; and then for each $(u, v, z) \in S_{u,v,\perp}$, delete (u, v, z) from S, pick an out-neighbor w of v uniformly at random, and add (v, u, w) to S.
- (for Case 3) retrieve $S_{u,\perp,v}$; and then for each $(u, y, v) \in S_{u,\perp,v}$ with $y \neq v$, delete (u, y, v) from S, pick an out-neighbor w of u uniformly at random, and add (u, y, w) to S.

By Lemma 7, $S_{u,v,\perp}$ and $S_{u,\perp,v}$ both have size $\tilde{O}(s/m)$ w.h.p. Thus, Cases 1-3 can be handled in $\tilde{O}(d(u) \cdot s/m))$ time w.h.p.

Next, we focus on Case 4. Let v be a neighbor of u with $\{u, v\}$ flipped. The number k_v of outneighbor-replaced tuples (x, y, z) with $x = v$ follows the binomial distribution $B(|S_{v,\perp,\perp}|, 1/d^+(v))$. Combining this with (the first bullet of) Lemma 7 shows that $k_v = \tilde{O}(d^+(v) \cdot \frac{s}{m} \cdot \frac{1}{d^+(v)}) = \tilde{O}(s/m)$ w.h.p. We extract a WoR sample set of size k_v from $S_{v,\perp,\perp}$[7], which takes $\tilde{O}(k_v) = \tilde{O}(s/m)$ time using the auxiliary structures. Every tuple (v, y, z) extracted is then modified to (v, y, u) in $\tilde{O}(1)$ time. Therefore, the total cost of Case 4 is again $\tilde{O}(d(u) \cdot s/m)$ w.h.p.

[7] Precisely speaking, this should be the $S_{v,\perp,\perp}$ at the beginning of `fix-D`(u).

Now, let us worry about the function D^+. Compared to before **fix-D**(u) was called, $d^+(u)$ may have changed abruptly (by as much as $d(u)$ in the worst case). If $D^+(u)$ now violates (5), we invoke **fix-Dplus**(u). Finally, for each neighbor v of u in G, $d^+(v)$ may have changed by 1, compared to before **fix-D**(u) was called. $D^+(v)$ may no longer satisfy (5); if so, call **fix-Dplus**(v).

In summary, **fix-D**(u) runs in $\tilde{O}(d(u) \cdot s/m)$ time w.h.p., plus the cost of all the calls to **fix-Dplus** at the end. It is worth pointing out that the invariant of **fix-D**(u) ensures $d(u) = O(\mathcal{D}_{old})$, where $\mathcal{D}_{old}$ is the value of $D(u)$ at the beginning of **fix-D**(u).

3.5 Analysis

Section 3.3 has shown that an insertion finishes in $\tilde{O}(s/m)$ time w.h.p. if no calls to **fix-Dplus** or **fix-D** are made. It remains to discuss the time spent on **fix-Dplus** and **fix-D**.

Let us start with **fix-D**. Consider its execution on a node u. Denote by $\mathcal{D}_{old}$ the value of $D(u)$ at the beginning of **fix-D**(u). Recall that **fix-D**(u) has cost $\tilde{O}(\mathcal{D}_{old} \cdot s/m)$ w.h.p., plus the cost of some calls to **fix-Dplus** at the end. We will account for the $\tilde{O}(\mathcal{D}_{old} \cdot s/m)$ cost first, and worry about **fix-Dplus** later. The invariant of **fix-D** (Section 3.4) makes sure that $\Omega(\mathcal{D}_{old})$ edges incident on u must have been inserted since the last time **fix-D** was invoked on u. We can therefore charge the $\tilde{O}(\mathcal{D}_{old} \cdot s/m)$ cost over those insertions, each of which bears only $\tilde{O}(s/m)$.

Let us now turn attention to **fix-Dplus**, for which we use a token-based analysis. A *token* is generated in two scenarios:

- Case 1: in Section 3.3, when an edge (u^*, v^*) is added to G^+, we give a token to u^* because its out-degree will increase by 1.
- Case 2: during the execution of **fix-D**(u), when we flip an in/out-edge of u with respect to an in/out-neighbor v, we give both u and v a token because their out-degrees will change by 1.

▶ **Lemma 10.** *If the total number of edge insertions is n_{ins}, the number of tokens generated is $O(n_{ins})$.*

Proof. The number of tokens in Case 1 is clearly n_{ins}. Next, we focus on Case 2. Let $\mathcal{D}_{old}$ be the value of $D(u)$ at the beginning of **fix-D**(u). Case 2 can generate at most $2d(u)$ tokens, while $2d(u)$ is $O(\mathcal{D}_{old})$ due to the invariant of **fix-D**. As mentioned, $\Omega(\mathcal{D}_{old})$ edges incident on u must have been inserted since the last **fix-D**(u). Thus, after amortization, each of those insertions generates $O(1)$ tokens in Case 2. ◀

Consider a call to **fix-Dplus**(u). Let $\mathcal{D}^+_{old}$ be the value of $D^+(u)$ at the beginning of the call. Clearly, u must have received at least $|\mathcal{D}^+_{old} - d^+(u)|$ tokens since the last **fix-Dplus**(u). We can charge the cost $\tilde{O}(|\mathcal{D}^+_{old} - d^+(u)| \cdot s/(\epsilon m))$ of **fix-Dplus**(u) over those tokens, each of which is amortized only $\tilde{O}(s/(\epsilon m))$. Combined with Lemma 10, this means that each insertion is amortized a share of $\tilde{O}(s/(\epsilon m))$.

In summary, each insertion runs in $\tilde{O}(s/(\epsilon m)) = \tilde{O}((1/\epsilon)^3 \cdot \sqrt{m}/\Gamma)$ amortized time w.h.p. This, together with the deletion algorithm in Appendix C, establishes Theorem 2.

3.6 Discussion

There is a rich literature on approximate triangle counting; for entry points into the literature, see [2–5, 9–12, 15, 18, 23–26, 30–32, 34, 35, 37]. The presented data structure reflects our efforts in identifying the existing techniques suitable for DACT. Strictly speaking, the "folklore"

static-counting algorithm in Section 3.1 has not been formally documented; however, its underlying ideas are already known. First, orienting the edges in the way described is a standard approach (e.g., [2, 4, 13, 16, 22, 31]). Second, the sampling procedure for acquiring "random tuples" is commonly known as *wedge sampling*, and is an important method behind many algorithms (e.g., [2,4,10,16,18,23,31,34,35]). Third, the notion of *contribution* (defined in (6)) is what makes wedge sampling work in our context, and was inspired by a subroutine inside an algorithm developed in [16] (see the `Heavy` subroutine therein). Our contributions, on the other hand, are in maintaining the information needed by the static algorithm under updates. The two-level approximation idea – manifested by the functions D and D^+ – is unlikely the only way to make things work, but has helped considerably in making our arguments as clean as possible.

4 A Structure for Dynamic Exact Triangle Counting

This section presents a DETC structure that achieves the performance in Theorem 3. Our algorithms and analysis can be regarded as a fine-grained version of those in [27].

4.1 Structure

We assume that the number m of edges in $G = (V, E)$ satisfies $M \leq m \leq 2M$ for some integer $M \geq 1$; the assumption can be removed by standard global rebuilding. As stated in Theorem 3, our structure takes a function $\Gamma(.)$ as a parameter. Set $\lambda = \Gamma(M)$ in the following discussion.

Graph orientation. At any moment, we orient G by giving each edge $\{u, v\}$ in G a direction. Let E^+ be the set of directed edges obtained, and denote by $G^+ = (V, E^+)$ the resulting directed graph. Denote by $d^+(u)$ the out-degree of $u \in V$. The orientation is done according to:

▶ **Lemma 11** ([6]). *By spending $O(\log m)$ worst-case time on an (edge) insertion/deletion in G, we can maintain G^+ such that $d^+(u) = O(\alpha + \log m)$ for every $u \in V$, where α is the largest arboricity of G in history. Furthermore, each insertion/deletion in G flips the directions of $O(\log m)$ edges in G^+. The above statements are true even if α is unknown.*

Since $M \leq m \leq 2M$ holds at all times, we must have $\alpha = O(\sqrt{M}) = O(\sqrt{m})$. Note that G^+ *can contain cycles* (it differs from the G^+ in Section 3.2). For each $u \in V$, denote by $N^+(u)$ the set of out-neighbors of u, and by $N^-(u)$ the set of its in-neighbors.

Light, heavy, and active H-combos. We classify each vertex $u \in V$ as *light* or *heavy* based on its degree $d(u)$ in G according to the rules below:

- if $d(u) \leq \lambda/2$, always light, whereas if $d(u) \geq \lambda$, always heavy;
- when our data structure is just constructed, u is heavy if $d(u) \geq 3\lambda/4$ or light otherwise;
- if u is heavy, it switches to light only when $d(u)$ has dropped to $\lambda/2$;
- if u is light, it switches to heavy only when $d(u)$ has increased to λ.

Given two distinct heavy vertices $u, v \in V$, define:

$$I_{\{u,v\}} = |N^-(u) \cap N^-(v)|$$

namely, the number of their common in-neighbors in G^+. $\{u, v\}$ forms an *active H-combo* if $I_{\{u,v\}} > 0$; note that there may not be an edge between u and v in G. Notice that while light/heavy-vertices are defined based on G, $I_{\{u,v\}}$ is defined based on G^+. This is a crucial design to attain the performance in Theorem 3.

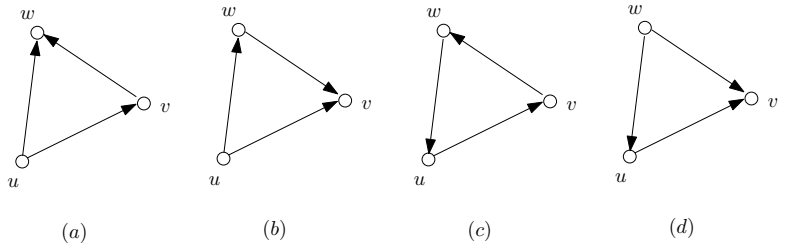


Figure 2 Four different triangles to be counted .

Structure. We maintain

- G and G^+
- the number T of triangles in G
- the set A of active H-combos, and $\mathcal{I}_A = \{I_{\{u,v\}} \mid \{u,v\} \in A\}$.

We also assume *auxiliary structures* for:

- given any heavy vertices u, v, finding $I_{\{u,v\}}$ in $\tilde{O}(1)$ time or declaring that $\{u,v\} \notin A$;
- inserting, deleting, or modifying an element in $\mathcal{I}_A$ using $\tilde{O}(1)$ time.

▶ **Lemma 12.** *The above structure consumes* $O(\min\{\alpha m + m \log m, (m/\lambda)^2\})$ *space.*

Proof. The auxiliary structures only need to be binary search trees which consume $O(|A|)$ space. It suffices to bound the size of A. Note that the number of heavy vertices is $O(m/\lambda)$ which immediately implies $|A| = O((m/\lambda)^2)$. Next, we will prove that $|A|$ is also bounded by $O(\alpha m + m \log m)$. Remember that each active H-combo $\{u,v\}$ must have a common in-neighbor. Conversely, each vertex $w \in V$ can generate $O(|N^+(w)|^2)$ active H-combos. By Lemma 11, $|N^+(w)| = O(\alpha + \log m)$. Therefore, $|A| = O(\sum_{w \in V} |N^+(w)|^2) = O(\sum_{w \in V} |N^+(w)| \cdot (\alpha + \log m)) = O(m(\alpha + \log m))$. ◀

Each (DETC) query obviously can be answered in constant time.

Remarks. For each edge update in G, Lemma 11 flips $O(\log m)$ edges in G^+. We implement the flipping of an edge (u,v) by first deleting (u,v) from G^+ and then adding (v,u) back. In this way, the number of edge updates on G^+ can be higher than that on G by at most a logarithmic factor. Thus, it suffices to discuss how to add/remove a (directed) edge in G^+.

Next, we will explain how to support insertions. The deletion algorithm is similar and thus moved to Appendix D. Our discussion will ignore the auxiliary structures. Furthermore, whenever an H-combo $\{u,v\}$ is inserted/deleted in A, $I_{\{u,v\}}$ can be inserted/deleted accordingly in logarithmic time. We will therefore not elaborate on the modifications to $\mathcal{I}_A$ caused by insertions/deletions in A.

4.2 Insertion

Update T. Given a new edge (u,v) in G^+, Figure 2 shows the possible cases for a triangle involving u and v, in terms of the edge directions. The types in Figures 2(a), 2(b), and 2(c) have an out-edge of u, v, or both, and hence, can be *enumerated* directly by scanning through the out-neighbors of u and v. The time required is $\tilde{O}(d^+(u) + d^+(v)) = \tilde{O}(\alpha)$ by Lemma 11.

Regarding Figure 2(d), we distinguish two cases:

- Case 1: u or v is a light vertex. If u (or v, resp.) is a light vertex, go through its $O(\lambda)$ in-edges to enumerate triangles of Figure 2(d) in $\tilde{O}(\lambda)$ time.

- Case 2: u and v are both heavy vertices. The number of such triangles is $I_{\{u,v\}}$, and can be retrieved from the auxiliary structures in $\tilde{O}(1)$ time.

Therefore, T can be updated in $\tilde{O}(\alpha + \lambda)$ time.

Update $\mathcal{I}_A$ and A. If v is heavy, every heavy out-neighbor w of u (other than v) forms an active H-combo with v. If $\{v, w\}$ is already in A, increase $I_{\{v,w\}}$ by 1; otherwise, add $\{v, w\}$ to A. This requires $\tilde{O}(d^+(u)) = \tilde{O}(\alpha)$ time in total.

Now, u and/or v may have just turned from light to heavy. It suffices to concentrate on u due to symmetry. We examine every in-neighbor x of u in G. For each heavy out-neighbor y of x ($y \neq u$), either add $\{u, y\}$ to A or increase $I_{\{u,y\}}$ by 1. The total time is $\tilde{O}(\alpha\lambda)$ because u has at most λ in-neighbors, each having an out-degree $\tilde{O}(\alpha)$. We charge the time on the $\Omega(\lambda)$ edges of u that have been added since u turned light last time; the insertion of each of those edges bears only $\tilde{O}(\alpha)$ time.

Combining the above with the deletion algorithm in Appendix D establishes Theorem 3.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

2 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *Innovations in Theoretical Computer Science (ITCS)*, pages 6:1–6:20, 2019.

3 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 2002.

4 Suman K. Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 11:1–11:14, 2017.

5 Suman K. Bera and C. Seshadhri. How the degeneracy helps for triangle counting in graph streams. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 457–467, 2020.

6 Edvin Berglin and Gerth Stølting Brodal. A simple greedy algorithm for dynamic graph orientation. *Algorithmica*, 82(2):245–259, 2020.

7 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 303–318, 2017.

8 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. *ACM Transactions on Database Systems (TODS)*, 43(2):7:1–7:32, 2018.

9 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 244–254, 2013.

10 Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. Triangle counting in dynamic graph streams. *Algorithmica*, 76(1):259–278, 2016.

11 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 253–262, 2006.

12 Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 7:1–7:18, 2020.

13 N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal of Computing*, 14(1):210–223, 1985.

14 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

15 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams (corrected). *Theoretical Computer Science*, 683:22–30, 2017.

16 Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM J. Comput.*, 46(5):1603–1646, 2017.

17 David Eppstein and Emma S. Spiro. The h-index of a graph and its application to dynamic subgraph statistics. *J. Graph Algorithms Appl.*, 16(2):543–567, 2012.

18 David García-Soriano and Konstantin Kutzkov. Triangle counting in streamed graphs via small vertex covers. In *SIAM International Conference on Data Mining (SDM)*, pages 352–360, 2014.

19 Isaac Goldstein, Moshe Lewenstein, and Ely Porat. On the hardness of set disjointness and set intersection with bounded universe. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 7:1–7:22, 2019.

20 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2015.

21 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. *CoRR*, abs/1511.06773, 2015.

22 Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. I/O-Efficient Algorithms on Triangle Listing and Counting. *ACM Transactions on Database Systems (TODS)*, 39(4):27:1–27:30, 2014.

23 Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of ACM Knowledge Discovery and Data Mining (SIGKDD)*, pages 589–597, 2013.

24 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics*, pages 710–716, 2005.

25 John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1778–1797, 2017.

26 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 598–609, 2012.

27 Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Counting triangles under updates in worst-case optimal time. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 4:1–4:18, 2019.

28 Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Dynamic set intersection. *CoRR*, abs/1407.6755, 2014.

29 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1272–1287, 2016.

30 Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. Approximate counting of cycles in streams. In *Proceedings of European Symposium on Algorithms (ESA)*, 2011.

31 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 401–411, 2016.

32 Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters (IPL)*, 112(7):277–281, 2012.

33 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 603–610, 2010.

34 A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment (PVLDB)*, 6(14):1870–1881, 2013.

35 C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Statistical Analysis and Data Mining*, 7(4):294–307, 2014.

36 Cheng Sheng, Yufei Tao, and Jianzhong Li. Exact and approximate algorithms for the most connected vertex problem. *ACM Transactions on Database Systems (TODS)*, 37(2):12:1–12:39, 2012.

37 Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15(6):703–726, 2011.

38 Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

Appendix

A Chernoff Bounds

Let $\mathcal{X}_1, ..., \mathcal{X}_n$ be independent random variables between 0 and 1. If $\mathcal{X} = \sum_{i=1}^{n} \mathcal{X}_i$ and $\mu = \mathbf{E}[\mathcal{X}]$, then for any $0 \leq \gamma \leq 1$:

$$\mathbf{Pr}\left[|\mathcal{X} - \mu| \geq \gamma \cdot \mu\right] \quad \leq \quad 2\exp\left(-\frac{\gamma^2\mu}{3}\right) \tag{9}$$

and for any $\gamma \geq 1$:

$$\mathbf{Pr}[\mathcal{X} \geq (1+\gamma)\cdot\mu] \quad \leq \quad \exp\left(-\frac{(1+\gamma)\mu}{6}\right). \tag{10}$$

These bounds can be found in [36].

B Proof of Lemma 8

We will use $N^+(u)$ to represent the set of out-neighbors of u in G^+.

▶ **Lemma 13.** *For every vertex $u \in V$, $d^+(u) \leq \max\{4, \sqrt{6m}\}$.*

Proof. We consider only $d(u) > 4$ because otherwise the claim obviously holds. For each out-neighbor v of u in G^+, its degree $d(v)$ in G must be at least 2. To see this, suppose on the contrary $d(v) \leq 1$, which implies $D(v) = 2 < \frac{d(u)}{2} \leq D(u)$ (the last $\leq$ is due to (4)). This means that the edge $\{u, v\}$ should point from v to u, giving a contradiction.

By (4), the fact $d(v) \geq 2$ indicates $D(v) \leq \frac{3}{2}d(v)$. We now have $\frac{d(u)}{2} \leq D(u) \leq D(v) \leq \frac{3d(v)}{2}$, namely, $d(u) \leq 3d(v)$. It follows that

$$d^+(u)^2 \leq d^+(u)\cdot d(u) = \sum_{v\in N^+(u)} d(u) \leq \sum_{v\in N^+(u)} 3d(v) \leq 6m$$

thus completing the proof. ◀

Let us introduce

$$D^+_{max} \quad = \quad (1+\epsilon/2)\cdot\max\{4, \sqrt{6m}\} \tag{11}$$

For each random tuple $(x, y, z) \in S$, define:

$$\mathcal{X}_{(x,y,z)} \quad = \quad \frac{f(x,y,z)}{D^+_{max}}. \tag{12}$$

Note that $\mathcal{X}_{(x,y,z)}$ is a random variable between 0 and 1 because $f(x,y,z) \leq D^+(x)$, while $D^+(x)$ is at most $(1+\epsilon/2) \cdot d^+(x)$ (see (5)), which in turn is at most D^+_{max} by Lemma 13. Set

$$\mathcal{X} = \sum_{(x,y,z)\in S} \mathcal{X}_{(x,y,z)} = \frac{\Lambda}{D^+_{max}} \tag{13}$$

where the last equality used (7).

▶ **Lemma 14.** $\left(1-\frac{\epsilon}{2}\right)\frac{s\cdot T}{m\cdot D^+_{max}} \leq \mathbf{E}[\mathcal{X}] \leq \left(1+\frac{\epsilon}{2}\right)\frac{s\cdot T}{m\cdot D^+_{max}}$.

Proof. On condition that (x,y) equals edge (u,v) in G^+, the random variable $f(x,y,z)$ takes value $D^+(u)$ if $(v,z) \in E^+$ or 0 otherwise. $(v,z) \in E^+$ if and only if z is a common out-neighbor of u and v. Hence:

$$\begin{aligned}
\mathbf{E}[f(x,y,z)] &= \frac{1}{m}\sum_{(u,v)\in E^+} \frac{|N^+(u)\cap N^+(v)|}{d^+(u)} \cdot D^+(u) \\
\text{(by (5))} &\leq \frac{1}{m}\sum_{(u,v)\in E^+} |N^+(u)\cap N^+(v)| \cdot (1+\epsilon/2) = (1+\epsilon/2)\cdot\frac{T}{m}.
\end{aligned}$$

It thus follows from (12) and (13) that $\mathbf{E}[\mathcal{X}] \leq (1+\epsilon/2)\frac{s\cdot T}{m\cdot D^+_{max}}$.

Analogously, applying the fact that $D^+(u)/d^+(u) \geq 1-\epsilon/2$ for all $u \in V$ leads to $\mathbf{E}[\mathcal{X}] \geq (1-\epsilon/2)\frac{s\cdot T}{m\cdot D^+_{max}}$. ◀

We will proceed differently from here, depending on the comparison between T and $\Gamma(m)$.

B.1 When $T \geq \Gamma(m)$

We will prove that $s = \tilde{O}((1/\epsilon)^2 \cdot M^{1.5}/\Gamma(M))$ ensures:

$$\mathbf{Pr}\left[\left|\Lambda\cdot\frac{m}{s} - T\right| \geq \epsilon\cdot T\right] \leq \frac{1}{m^3} \tag{14}$$

▶ **Lemma 15.** *We can choose an $s = \tilde{O}((1/\epsilon)^2 \cdot M^{1.5}/\Gamma(M))$ to guarantee*

$$\mathbf{Pr}\left[|\mathcal{X} - \mathbf{E}[\mathcal{X}]| \geq \frac{\epsilon}{2}\cdot\frac{s\cdot T}{m\cdot D^+_{max}}\right] \leq \frac{1}{m^3}.$$

Proof.

$$\begin{aligned}
& \mathbf{Pr}\left[|\mathcal{X} - \mathbf{E}[\mathcal{X}]| \geq \frac{\epsilon}{2}\cdot\frac{s\cdot T}{m\cdot D^+_{max}}\right] \\
\text{(by Lemma 14)} \quad \leq & \mathbf{Pr}\left[|\mathcal{X} - \mathbf{E}[\mathcal{X}]| \geq \frac{\epsilon}{2}\cdot\frac{\mathbf{E}[X]}{1+\epsilon/2}\right] \leq \mathbf{Pr}\left[|\mathcal{X} - \mathbf{E}[\mathcal{X}]| \geq \frac{\epsilon}{3}\cdot\mathbf{E}[X]\right] \\
\text{(by (9))} \quad \leq & 2\exp\left(-\left(\frac{\epsilon}{3}\right)^2\frac{\mathbf{E}[\mathcal{X}]}{3}\right) \\
\text{(by Lemma 14)} \quad \leq & 2\exp\left(-\left(\frac{\epsilon}{3}\right)^2\frac{s\cdot T}{3m\cdot D^+_{max}}\cdot(1-\epsilon/2)\right) \\
\leq & 2\exp\left(-\left(\frac{\epsilon}{3}\right)^2\frac{s\cdot T}{6m\cdot D^+_{max}}\right)
\end{aligned}$$

which is at most $1/m^3$ for $s = O(\frac{mD^+_{max}}{\epsilon^2\cdot T}\log m)$. The claim follows from $D^+_{max} = O(\sqrt{m})$ (see (11)), $T \geq \Gamma(m) \geq \Gamma(M)$, and $m \leq 2M$. ◀

The lemma implies (14) because

$$\begin{aligned}
& \mathbf{Pr}\left[\left|\mathcal{X}-\mathbf{E}[\mathcal{X}]\right| \geq \frac{\epsilon}{2}\cdot\frac{s\cdot T}{m\cdot D^{+}_{max}}\right] \\
= \;& \mathbf{Pr}\left[\mathcal{X}\geq \mathbf{E}[\mathcal{X}]+\frac{\epsilon}{2}\cdot\frac{s\cdot T}{m\cdot D^{+}_{max}} \text{ or } \mathcal{X}\leq \mathbf{E}[\mathcal{X}]-\frac{\epsilon}{2}\cdot\frac{s\cdot T}{m\cdot D^{+}_{max}}\right] \\
(\text{by Lemma 14}) \;\geq\; & \mathbf{Pr}\left[\mathcal{X}\geq (1+\epsilon)\frac{s\cdot T}{m\cdot D^{+}_{max}} \text{ or } \mathcal{X}\leq (1-\epsilon)\frac{s\cdot T}{m\cdot D^{+}_{max}}\right] \\
(\text{by (13)}) \;=\; & \mathbf{Pr}\left[\Lambda\cdot\frac{m}{s}\geq (1+\epsilon)T \text{ or } \Lambda\cdot\frac{m}{s}\leq (1-\epsilon)T\right] \\
= \;& \mathbf{Pr}\left[\left|\Lambda\cdot\frac{m}{s}-T\right|\geq \epsilon\cdot T\right].
\end{aligned}$$

B.2 When $0 < T < \Gamma(m)$

We will prove that $s = \tilde{O}((1/\epsilon)^2\cdot M^{1.5}/\Gamma(M))$ ensures:

$$\mathbf{Pr}\left[\left|\Lambda\cdot\frac{m}{s}-T\right|\geq \epsilon\cdot\Gamma(m)\right] \;\leq\; \frac{1}{m^3} \tag{15}$$

Since there is at least one triangle, $\mathcal{X}_{x,y,z}$ (see (12)) has expectation strictly greater than 0. By (13), this means $\mathbf{E}[\mathcal{X}] > 0$.

▶ **Lemma 16.** *We can choose an $s = \tilde{O}((1/\epsilon)^2\cdot M^{1.5}/\Gamma(M))$ to guarantee*

$$\mathbf{Pr}\left[\left|\mathcal{X}-\mathbf{E}[\mathcal{X}]\right|\geq\frac{\epsilon}{2}\cdot\frac{s\cdot\Gamma(m)}{m\cdot D^{+}_{max}}\right] \;\leq\; \frac{1}{m^3}.$$

Proof.

$$\mathbf{Pr}\left[\left|\mathcal{X}-\mathbf{E}[\mathcal{X}]\right|\geq\frac{\epsilon}{2}\cdot\frac{s\cdot\Gamma(m)}{m\cdot D^{+}_{max}}\right] = \mathbf{Pr}\left[\left|\mathcal{X}-\mathbf{E}[\mathcal{X}]\right|\geq\frac{\epsilon}{2}\cdot\frac{s\cdot\Gamma(m)}{m\cdot D^{+}_{max}\cdot\mathbf{E}[\mathcal{X}]}\cdot\mathbf{E}[\mathcal{X}]\right] \tag{16}$$

Setting $\gamma = \frac{\epsilon\cdot s\cdot\Gamma(m)}{2m\cdot D^{+}_{max}\cdot\mathbf{E}[\mathcal{X}]}$, we distinguish two cases.

Case 1: $\gamma \leq 1$. By (9), we have

$$\begin{aligned}
(16) \;&\leq\; 2\exp\left(-\gamma^2\cdot\frac{\mathbf{E}[\mathcal{X}]}{3}\right) = 2\exp\left(-\left(\frac{\epsilon\cdot s\cdot\Gamma(m)}{2m\cdot D^{+}_{max}}\right)^2\cdot\frac{1}{3\,\mathbf{E}[\mathcal{X}]}\right) \\
(\text{by Lemma 14}) \;&\leq\; 2\exp\left(-\left(\frac{\epsilon\cdot s\cdot\Gamma(m)}{2m\cdot D^{+}_{max}}\right)^2\cdot\frac{1}{3(1+\epsilon/2)\frac{s\cdot T}{m\cdot D^{+}_{max}}}\right) \\
&\leq\; 2\exp\left(-\frac{\epsilon^2 s\cdot(\Gamma(m))^2}{18m\cdot T\cdot D^{+}_{max}}\right) \\
(\text{by } \Gamma(m) > T) \;&\leq\; 2\exp\left(-\frac{\epsilon^2 s\cdot\Gamma(m)}{18m\cdot D^{+}_{max}}\right)
\end{aligned}$$

which is at most $1/m^3$ for $s = O(\frac{mD^{+}_{max}\cdot\log m}{\epsilon^2\Gamma(m)})$. The claim follows from $D^{+}_{max} = O(\sqrt{m})$, $\Gamma(m)\geq\Gamma(M)$, and $m\leq 2M$.

Case 2: $\gamma > 1$. Since $\mathcal{X} \geq 0$, we have

$$\begin{aligned}
(16) &= \mathbf{Pr}\Big[\mathcal{X} - \mathbf{E}[\mathcal{X}] \geq \gamma \cdot \mathbf{E}[\mathcal{X}]\Big] \\
(\text{by (10)}) &\leq \exp\left(-\frac{1+\gamma}{6}\,\mathbf{E}[\mathcal{X}]\right) \leq \exp\left(-\frac{\gamma}{6}\,\mathbf{E}[\mathcal{X}]\right) \\
&\leq \exp\left(-\frac{\epsilon \cdot s \cdot \Gamma(m)}{12m \cdot D^+_{max} \cdot \mathbf{E}[\mathcal{X}]} \cdot \mathbf{E}[\mathcal{X}]\right) = \exp\left(-\frac{\epsilon \cdot s \cdot \Gamma(m)}{12m \cdot D^+_{max}}\right)
\end{aligned}$$

which is at most $1/m^3$ for $s = O(\frac{mD^+_{max} \cdot \log m}{\epsilon \cdot \Gamma(m)})$. The claim follows from $D^+_{max} = O(\sqrt{m})$, $\Gamma(m) \geq \Gamma(M)$, and $m \leq 2M$. ◀

The lemma implies (15) because

$$\begin{aligned}
&\mathbf{Pr}\left[\left|\mathcal{X} - \mathbf{E}[\mathcal{X}]\right| \geq \frac{\epsilon}{2} \cdot \frac{s \cdot \Gamma(m)}{m \cdot D^+_{max}}\right] \\
= \;&\mathbf{Pr}\left[\mathcal{X} \geq \mathbf{E}[\mathcal{X}] + \frac{\epsilon}{2} \cdot \frac{s \cdot \Gamma(m)}{m \cdot D^+_{max}} \text{ or } \mathcal{X} \leq \mathbf{E}[\mathcal{X}] - \frac{\epsilon}{2} \cdot \frac{s \cdot \Gamma(m)}{m \cdot D^+_{max}}\right] \\
(\text{by Lemma 14}) \geq \;&\mathbf{Pr}\Big[\mathcal{X} \geq \left(1 + \frac{\epsilon}{2}\right) \frac{s \cdot T}{mD^+_{max}} + \frac{\epsilon s \cdot \Gamma(m)}{2mD^+_{max}} \text{ or} \\
&\qquad \mathcal{X} \leq \left(1 - \frac{\epsilon}{2}\right) \frac{s \cdot T}{mD^+_{max}} - \frac{\epsilon s \cdot \Gamma(m)}{2mD^+_{max}}\Big] \\
(\text{by } T < \Gamma(m)) \geq \;&\mathbf{Pr}\left[\mathcal{X} \geq \frac{s \cdot T}{m \cdot D^+_{max}} + \frac{\epsilon s \cdot \Gamma(m)}{m \cdot D^+_{max}} \text{ or } \mathcal{X} \leq \frac{s \cdot T}{m \cdot D^+_{max}} - \frac{\epsilon s \cdot \Gamma(m)}{m \cdot D^+_{max}}\right] \\
(\text{by (13)}) = \;&\mathbf{Pr}\left[\Lambda \cdot \frac{m}{s} \geq T + \epsilon \cdot \Gamma(m) \text{ or } \Lambda \cdot \frac{m}{s} \leq T - \epsilon \cdot \Gamma(m)\right] \\
= \;&\mathbf{Pr}\left[\left|\Lambda \cdot \frac{m}{s} - T\right| \geq \epsilon \cdot \Gamma(m)\right].
\end{aligned}$$

B.3 When $T = 0$

In this case, every random tuple must have contribution 0 (see (6)). Thus, Λ must be 0, and hence, so is our estimate.

C Deletion Algorithm of the DATC Structure

C.1 Deletion: When D Will Still Be Good

Suppose that we are deleting an edge $\{u^*, v^*\}$ in G. This section discusses the scenario where D is still good after $d(u^*)$ and $d(v^*)$ decrease by 1. Assume, without loss of generality, that $\{u^*, v^*\}$ points from u^* to v^* in G^+.

Every $(x, y, z) \in S$ with $x = u^*$ and $y = v^*$ should be replaced with a new random tuple. For this purpose, we remove the entire $S_{(u^*,v^*,\perp)}$ from S, regenerate the same the same number of random tuples, and add them to S. By Lemma 7, this can be done in $\tilde{O}(s/m)$ time w.h.p.

Now consider a tuple $(x, y, z) \in S$ with $x \neq u^*$ or $y \neq v^*$. After the deletion, (x, y) remains as a uniformly random edge in from E^+. Nevertheless, we still need to make sure that z is a random out-neighbor of x, and that Λ is correct:

- Case 1: $x = u^*$ and $z = v^*$. We remove (x, y, z) from S, select an out-neighbor w of u^* uniformly at random, and add (u^*, y, w) to S.

- Case 2: $y = u^*$ and $z = v^*$. As the deleted edge (u^*, v^*) breaks the triangle formed by x, u^*, and v^*, Λ should be decreased by $D^+(x)$.

Regarding implementation, all the tuples of Case 1 can be found in $\tilde{O}(|S_{(u^*,\perp,v^*)}|)$ time, which is $\tilde{O}(s/m)$ w.h.p. by Lemma 7; this is also the time spent on Case 1 in total. For Case 2, the overall amount of reduction on Λ (summing up over all tuples of Case 2) is simply Ξ_{u^*,v^*}, which can be retrieved in $\tilde{O}(1)$ time; and then Λ can be adjusted in constant time.

Finally, if $D^+(u^*)$ no longer satisfies (5), we simply call `fix-Dplus`(u^*) (Section 3.3).

In summary, the deletion runs in $\tilde{O}(s/m)$ time w.h.p, plus the cost of at most one call to `fix-Dplus`.

C.2 Deletion: When D Will Go Bad

We now consider the scenario where D violates (4) after $\{u^*, v^*\}$ is deleted. Similar to Section 3.4, we reduce the case to Section C.1 by first modifying D such that it will still be good after the deletion. Due to symmetry, it suffices to discuss only the situation where $D(u^*)$ needs to be fixed.

The fix is performed by `fix-D-del`(u), which has the constraint:

Invariant: when called:
- D is good
- $d(u) < D(u)$ and
- $D(u) - d(u) = \Omega(D(u))$.

At the end of `fix-D-del`(u), $D(u) = d(u)$. It is rudimentary to verify that D will still be good after $d(u)$ drops by 1.

Rationale behind `fix-D-del`(u). We decrease $D(u)$ to $d(u)$, which may affect the direction of an edge in G^+ incident on u: if the direction was (v, u) before, it may now be flipped to (u, v).

Fix a tuple $(x, y, z) \in S$. Consider an arbitrary edge (v, u) that has been flipped to (u, v). The next discussion clarifies all the cases that require modifications:

- Case 1: $x \neq v$ and $x \neq u$. (x, y, z) still remains as a random tuple, but its contribution may change:
  - Case 1.1: $y = u$ and $z = v$. $f(x, y, z)$ will grow from 0 to $D^+(x)$. Accordingly, Λ needs to be increased by $D^+(x)$.
  - Case 1.2: $y = v$ and $z = u$. $f(x, y, z)$ will drop from $D^+(x)$ to 0. Accordingly, Λ needs to be decreased by $D^+(x)$.
- Case 2: $x = v$ and $y = u$. The tuple (v, u, z) should be replaced by (u, v, w) where w is a (uniformly) random out-neighbor of u.
- Case 3: $x = v$, $y \neq u$, and $z = u$. The tuple (v, y, u) should be replaced by (v, y, w) where w is a (uniformly) random out-neighbor of v.
- Case 4: $x = u$ (which implies $y \neq v$ and $z \neq v$). z should be replaced by v with probability $1/d^+(u)$. If the replacement occurs, the tuple (u, y, z) is said to be *outneighbor-replaced*.

Note the similarity to the cases in Section 3.4.

Algorithm `fix-D-del`(u). Set $D(u) = d(u)$ and flip the edges of u in G^+ wherever needed.

Given each neighbor v of u such that $\{u, v\}$ was flipped, we

- (for Case 1) retrieve $\Xi_{u,v}$ and $\Xi_{v,u}$ (from the auxiliary structures), and increase Λ by $\Xi_{u,v} - \Xi_{v,u}$.

- (for Case 2) retrieve $S_{v,u,\perp}$; and then for each $(v,u,z) \in S_{v,u,\perp}$, delete (v,u,z) from S, pick an out-neighbor w of u uniformly at random, and add (u,v,w) to S.
- (for Case 3) retrieve $S_{v,\perp,u}$; and then for each $(v,y,u) \in S_{v,\perp,u}$ with $y \neq u$, delete (v,y,u) from S, pick an out-neighbor w of v uniformly at random, and add (v,y,w) to S.

Case 1 obviously takes $\tilde{O}(d(u) \cdot s/m))$ time w.h.p. By Lemma 7, Cases 2 and 3 can also be handled in the same cost.

Next, we attend to Case 4. Consider any neighbor v of u with $\{u,v\}$ flipped. The number k_u of outneighbor-replaced tuples (x,y,z) with $x = u$ follows the binomial distribution $B(|S_{u,\perp,\perp}|, 1/d^+(u))$. This, together with Lemma 7, shows that $k_u = \tilde{O}(d^+(u) \cdot \frac{s}{m} \cdot \frac{1}{d^+(u)}) = \tilde{O}(s/m)$ w.h.p. We draw a WoR sample set of size k_u from $|S_{u,\perp,\perp}|$ in $\tilde{O}(k_u) = \tilde{O}(s/m)$ time. Every tuple (u,y,z) drawn is modified to (u,y,v) in $\tilde{O}(1)$ time. The total cost of Case 4 is $\tilde{O}(d(u) \cdot s/m)$ w.h.p.

Finally, if D^+ is bad, we remedy it in the same way as in Section 3.4.

In summary, `fix-D-del`(u) runs in $\tilde{O}(d(u) \cdot s/m)$ time w.h.p., plus the cost of all the calls to `fix-Dplus` at the end. The invariant ensures that $d(u) < \mathcal{D}_{old}$ where $\mathcal{D}_{old}$ is the value of $D(u)$ at the beginning of `fix-D-del`(u).

C.3 Analysis

The analysis is a straightforward adaptation of the argument in Section 3.5. It suffices to point out some key changes:

- The invariant of `fix-D-del` makes sure that $\Omega(\mathcal{D}_{old})$ edges incident on u have been removed since the last call to `fix-D-del`(u), where $\mathcal{D}_{old}$ is the value of $D(u)$ at the beginning of `fix-D-del`(u).
- When an edge (u^*, v^*) is deleted from G, we give u^* a token.
- During the execution of `fix-D-del`(u), when we flip an in/out-edge of u with respect an its in/out-neighbor v, we give a token to both u and v.
- Lemma 10 should be replaced with: if the total number of edge insertions/deletions is n_{upd}, the number of tokens generated is $O(n_{upd})$.

D Deletion Algorithm of the DETC Structure

Update T. Suppose that we are deleting (u,v) from G^+. The possible cases for a triangle involving u and v are the same as in Figure 2. The number of such triangles can be found in the same manner as in the insertion algorithm using $\tilde{O}(\alpha + \lambda)$ time. After that, T is updated in constant time.

Update $\mathcal{I}_A$ and A. If v is heavy, for every heavy out-neighbor $w \neq v$ of u, we decrease $I_{\{v,w\}}$ by 1. If $I_{\{v,w\}} = 0$, $\{v,w\}$ is removed from A. The time is $\tilde{O}(d^+(u)) = \tilde{O}(\alpha)$.

Vertex u (the case of v is similar) may have just turned from heavy to light. We examine every in-neighbor x of u in G. For each heavy out-neighbor y of x, remove $\{u,y\}$ from A. This takes $\tilde{O}(\alpha\lambda)$ time in total. We charge the time on the $\Omega(\lambda)$ edges of u that have been removed since u turned heavy last time. After amortization, the deletion of each of those edges bears only $\tilde{O}(\alpha)$ time.

We conclude that the deletion time is $\tilde{O}(\alpha + \lambda)$ amortized.

Grammars for Document Spanners

Liat Peterfreund ✉
DI ENS, ENS, CNRS, PSL University, Paris, France
Inria, Paris, France

Abstract

We propose a new grammar-based language for defining information-extractors from documents (text) that is built upon the well-studied framework of document spanners for extracting structured data from text. While previously studied formalisms for document spanners are mainly based on regular expressions, we use an extension of context-free grammars, called extraction grammars, to define the new class of context-free spanners. Extraction grammars are simply context-free grammars extended with variables that capture interval positions of the document, namely spans. While regular expressions are efficient for tokenizing and tagging, context-free grammars are also efficient for capturing structural properties. Indeed, we show that context-free spanners are strictly more expressive than their regular counterparts. We reason about the expressive power of our new class and present a pushdown-automata model that captures it. We show that extraction grammars can be evaluated with polynomial data complexity. Nevertheless, as the degree of the polynomial depends on the query, we present an enumeration algorithm for unambiguous extraction grammars that, after quintic preprocessing, outputs the results sequentially, without repetitions, with a constant delay between every two consecutive ones.

2012 ACM Subject Classification Information systems → Information extraction; Information systems → Relational database model; Information systems → Data model extensions

Keywords and phrases Information Extraction, Document Spanners, Context-Free Grammars, Constant-Delay Enumeration, Regular Expressions, Pushdown Automata

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.7

Related Version *Full Version*: `https://arxiv.org/abs/2003.06880`

Funding *Liat Peterfreund*: A part of this work was done while affiliated with IRIF – CNRS & Université de Paris, Paris, France, and with University of Edinburgh, Edinburgh, UK. This work is supported by the Fondation des Sciences Mathématiques de Paris (FSMP).

Acknowledgements I would like to thank the anonymous reviewers for their extremely valuable comments, and in particular those that enabled me to extend the enumeration algorithm to a broader class of extraction grammars. I am grateful to Arnaud Durand, Michael Kaminski, Benny Kimelfeld, and Leonid Libkin for useful discussions, and to Dominik D. Freydenberger for references.

1 Introduction

The abundance and availability of valuable textual resources in the last decades position text analytics as a standard component in data-driven workflows. One of the core operations that aims to facilitate the analysis and integration of textual content is Information Extraction (IE), the extraction of structured data from text. IE arises in a large variety of domains, including social media analysis [4], health-care analysis [43], customer relationship management [1], information retrieval [45], and more.

Rules have always been a key component in various paradigms for IE, and their roles have varied and evolved over the time. Systems such as Xlog [38] and IBM's SystemT [27, 6] use rules to extract relations from text (e.g., tokenizer, dictionary lookup, and part-of-speech tagger) that are further manipulated with relational query languages. Other systems use rules to generate features for machine-learning classifiers [26, 35].

x	y
$[1,2\rangle$	$[2,3\rangle$
$[3,4\rangle$	$[4,5\rangle$
$[3,4\rangle$	$[4,6\rangle$

Figure 1 Extracted relation.

$$S \to B \vdash_x \mathtt{a}A\mathtt{b} \dashv_y B$$
$$A \to \mathtt{a}A\mathtt{b} \mid \dashv_x\vdash_y$$
$$B \to \mathtt{a}B \mid \mathtt{b}B \mid \epsilon$$

Figure 2 Production rules.

$$\vdash_x \mathtt{aa} \dashv_x\vdash_y \mathtt{bb} \dashv_y \mathtt{b}$$
$$\mathtt{aa} \vdash_x \mathtt{aa} \dashv_x\vdash_y \mathtt{bb} \dashv_y \mathtt{b}$$
$$\mathtt{aa} \vdash_x \mathtt{a} \dashv_x\vdash_y \mathtt{b} \dashv_y \mathtt{b}$$

Figure 3 Ref-words.

Document Spanners. The framework of document spanners, presented by Fagin et al., provides a theoretical basis for investigating the principles of relational rule systems for IE [11]. The research on document spanners has focused on their expressive power [11, 14, 34, 17, 32, 19] their computational complexity [2, 13, 18, 33], incompleteness [29, 33], and other system aspects such as cleaning [12], dynamic complexity [20], distributivity [7] and an annotated variant [8].

In the documents spanners framework, a *document* $\mathbf{d}$ is a string over a fixed finite alphabet, and a *spanner* is a function that extracts from a document a relation over the spans of $\mathbf{d}$. A *span* x is a half-open interval of positions of $\mathbf{d}$ and it represents a substring $\mathbf{d}_x$ of $\mathbf{d}$ that is identified by these positions. A natural way to specify a spanner is by a *regex formula*: a regular expression with embedded *capture variables* that are viewed as relational attributes. For instance, the spanner that is given by the regex formula $(\mathtt{a} \vee \mathtt{b})^* \vdash_x \mathtt{aa}^* \dashv_x\vdash_y \mathtt{bb}^* \dashv_y (\mathtt{a} \vee \mathtt{b})^*$ extracts from documents spans x and y that correspond, respectively, with a non-empty substring of `a`'s followed by a non-empty substring of `b`'s. In particular, it extracts from the document `ababb` the relation depicted in Figure 1.

The class of *regular spanners* is the class of spanners definable as the closure of regex formulas under positive relational algebra operations: projection, natural join and union. The class of regular spanners can be represented alternatively by finite state machines, namely *variable-set automata (vset-automata)*, which are nondeterministic finite-state automata that can open and close variables (that, as in the case of regex formulas, play the role of the attributes of the extracted relation). *Core* spanners [11] are obtained by extending the class of regular spanners with string-equality selection on span variables. Although core spanners can express strictly more than regular spanners, they are still quite limited as, e.g., there is no core spanner that extracts all pairs x and y of spans having the same *length* [11].

To date, most research on spanners has been focused on the regular representation, that is, regular expressions and finite state automata. While regular expressions are useful for segmentation and tokenization, they are not useful in describing complex nested structures (e.g., syntactic structure of a natural language sentence) and relations between different parts of the text. Regular languages also fall short in dealing with tasks such as syntax highlighting [30] and finding patterns in source code [39]. For all of the above mentioned tasks we have context-free grammars. It is well known that context-free languages are strictly more expressive than regular languages. Büchi [5] has showed that regular languages are equivalent to monadic second order logic (over strings), and Lautemann et al. [25] have showed that adding an existential quantification over a binary relation interpreted as a matching is enough to express all context-free languages. This quantification, intuitively, is what makes it possible to also express structural properties.

Contribution. In this work we propose a new grammar-based approach for defining the class of *context-free spanners*. Context-free spanners are defined via *extraction grammars* which, like regex formulas, incorporate *capture variables* that are viewed as relational attributes.

Extraction grammars produce *ref-words* which are words over an extended alphabet that consists of standard terminal symbols along with *variable operations* that denote opening and closing of variables. The result of evaluating an extraction grammar on a document $\mathbf{d}$ is defined via the ref-words that are produced by the grammar and equal to $\mathbf{d}$ after erasing the variable operations. For example, the extraction grammar from Figure 2 produces also the ref-words $\vdash_x \mathtt{a} \dashv_x \vdash_y \mathtt{b} \dashv_y \mathtt{abb}$ and $\mathtt{ab} \vdash_x \mathtt{a} \dashv_x \vdash_y \mathtt{b} \dashv_y \mathtt{b}$. Hence, it extracts from $\mathbf{d} := \mathtt{ababb}$ the two first tuples from the relation in Figure 1. In Figure 3 there are additional examples of ref-words produced by this grammar. In general, the given grammar extracts from documents the spans x and y that correspond, respectively, with a non-empty substring of $\mathtt{a}$'s followed by an equal-length substring of $\mathtt{b}$'s. With a slight adaptation of Fagin et al. inexpressibility proof [11, Theorem 4.21], it can show that this spanner is inexpressible by core spanners.

Indeed, we show that context-free spanners are strictly more expressive than regular spanners and that the restricted class of regular extraction grammars captures the regular spanners. We compare the expressiveness of context-free spanners against core and generalized core spanners and show that context-free spanners are incomparable to any of these classes. In addition to extraction grammars, we present a pushdown automata model that captures the context-free spanners.

In term of evaluation of context-free spanners we can evaluate extraction grammars in polynomial time in *data complexity*, where the spanner is regarded as fixed and the document as input. However, as the degree of this polynomial depends on the query (in particular, on the number of variables in the relation it extracts), we propose an enumeration algorithm for unambiguous extraction grammars. Our algorithm outputs the results consecutively, after quintic preprocessing, with constant delay between every two answers. In the first step of the preprocessing stage we manipulate the extraction grammar so that it will be adjusted to the input document. In the second step of the preprocessing we change it in a way that its non-terminals include extra information on the variable operations. This extra information enables us to skip sequences of productions that do not affect the output, hence obtaining a delay that is independent of the input document and linear in the number of variables associated with the spanner.

Related Work. Grammar-based parsers are widely used in IE systems [44, 37]. There are, as well, several theoretical frameworks that use grammars for IE, one of which is Knuth's framework of attribute grammars [23, 24]. In this framework, the non-terminals of a grammar are attached with attributes[1] that pass semantic information up and down a parse-tree. While both extraction grammars and attribute grammars extract information via grammars, it seems as if the expressiveness of these formalisms is incomparable to extraction grammars.

The problem of enumerating words of context-free grammars arises in different contexts [41, 31]. Providing complexity guarantees on the enumeration is usually tricky and requires assumptions either on the grammar or on the output. Mäkinen [28] has presented an enumeration algorithm for regular grammars and for unambiguous context-free grammars with additional restrictions (strongly prefix-free and length complete). Later, Dömösi [9] has presented an enumeration algorithm for unambiguous context-free grammars that outputs, with quadratic delay, only the words of a fixed length.

Organization. In Section 2, we present extraction grammars and extraction pushdown automata. In Section 3, we discuss the expressive power of context-free spanners and their evaluation. In Sections 4 and 5, we present our enumeration algorithm, and in Section 6 we conclude.

[1] The term "attributes" was previously used in the relational context; Here the meaning is different.

2 Context-Free Spanners

In this section we present the class of context-free spanners by presenting two formalisms for expressing them: extraction grammars and extraction pushdown automata.

2.1 Preliminaries

We start by presenting the formal setup based on notations and definitions used in previous works on document spanners (e.g., [11, 18]).

Strings and Spans. We set an infinite set Vars of variables and fix a finite alphabet Σ that is disjoint of Vars. In what follows we assume that our alphabet Σ consists of at least two letters. A *document* $\mathbf{d}$ is a finite sequence over Σ whose length is denoted by $|\mathbf{d}|$. A *span* identifies a substring of $\mathbf{d}$ by specifying its bounding indices. Formally, if $\mathbf{d} = \sigma_1 \cdots \sigma_n$ where $\sigma_i \in \Sigma$ then a span of $\mathbf{d}$ has the form $[i, j\rangle$ where $1 \leq i \leq j \leq n+1$ and $\mathbf{d}_{[i,j\rangle}$ denotes the substring $\sigma_i \cdots \sigma_{j-1}$. When $i = j$ it holds that $\mathbf{d}_{[i,j\rangle}$ equals the empty string, which we denote by ϵ. We denote by $\mathsf{Spans}(\mathbf{d})$ the set of all possible spans of a document $\mathbf{d}$.

Document Spanners. Let $X \subseteq \mathsf{Vars}$ be a finite set of variables and let $\mathbf{d}$ be a document. An $(X, \mathbf{d})$-mapping assigns spans of $\mathbf{d}$ to variables in X. An $(X, \mathbf{d})$-relation is a finite set of $(X, \mathbf{d})$-mappings. A *document spanner* (or *spanner*, for short) is a function associated with a finite set X of variables that maps documents $\mathbf{d}$ into $(X, \mathbf{d})$-relations.

2.2 Extraction Grammars

The *variable operations* of a variable $x \in \mathsf{Vars}$ are $\vdash_x$ and $\dashv_x$ where, intuitively, $\vdash_x$ denotes the opening of x, and $\dashv_x$ its closing. For a finite subset $X \subseteq \mathsf{Vars}$, we define the set $\Gamma_X := \{\vdash_x, \dashv_x \mid x \in X\}$. That is, Γ_X is the set that consists of all the variable operations of all variables in X. We assume that Σ and Γ_X are disjoint. We extend the classical definition of context-free grammars [22] by treating the variable operations as special terminal symbols. Formally, a *context-free extraction grammar*, or *extraction grammar* for short, is a tuple $G := (X, V, \Sigma, P, S)$ where

- $X \subseteq \mathsf{Vars}$ is a finite set of variables,
- V is a finite set of *non-terminal* symbols[2],
- Σ is a finite set of *terminal* symbols;
- P is a finite set of *production rules* of the form $A \to \alpha$ where A is a non-terminal and $\alpha \in (V \cup \Sigma \cup \Gamma_X)^*$, and
- S is a designated non-terminal symbol referred to as the *start symbol*.

We say that the extraction grammar G is *associated* with X.

▶ **Example 1.** In this and in the following examples we often denote the elements in V by upper case alphabet letters from the beginning of the English alphabet $(A, B, C, \ldots)$. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, and let us consider the grammar DISJEQLEN associated with the variables $\{x, y\}$ that is given by the following production rules:

- $S \to B \vdash_x A \dashv_y B \mid B \vdash_y A \dashv_x B$
- $A \to \mathtt{a}A\mathtt{a} \mid \mathtt{a}A\mathtt{b} \mid \mathtt{b}A\mathtt{b} \mid \mathtt{b}A\mathtt{a}$

[2] Note that these are often referred to as variables, however, here we use the term "non-terminals" to distinguish between these symbols and elements in Vars.

- $A \to \dashv_x B \vdash_y \mid \dashv_y B \vdash_x$
- $B \to \epsilon \mid \mathtt{a}B \mid \mathtt{b}B$

Here and in what follows, we use the compact notation for production rules by writing $A \to \alpha_1 \mid \cdots \mid \alpha_n$ instead of the productions $A \to \alpha_1, \cdots, A \to \alpha_n$. As we shall later see, this grammar extracts pairs of disjoint spans with the same length. ⌟

While classical context-free grammars generate words, extraction grammars generate words over the extended alphabet $\Sigma \cup \Gamma_X$. These words are referred to as *ref-words* [36]. Similarly to (classical) context-free grammars, the process of deriving ref-words is defined via the notations $\Rightarrow, \Rightarrow^n, \Rightarrow^*$ that stand for one, n, and several (possibly zero) derivation steps, respectively. To emphasize the grammar being discussed, we sometime use the grammar as a subscript (e.g., $\Rightarrow^*_G$). For the full definitions we refer the reader to Hopcroft et al. [22]. A non-terminal A is called *useful* if there is some derivation of the form $S \Rightarrow^* \alpha A \beta \Rightarrow^* w$ where $w \in (\Sigma \cup \Gamma_X)^*$. If A is not useful then it is called *useless*. For complexity analysis, we define the *size* $|G|$ of an extraction grammar G as the sum of the number of symbols at the right-hand sides (i.e., to the right of $\to$) of its rules.

2.3 Semantics of Extraction Grammars

Following Freydenberger [15] we define the semantics of extraction grammars using ref-words. A ref-word $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ is *valid (for X)* if each variable of X is opened and then closed exactly once, or more formally, for each $x \in X$ the string $\mathbf{r}$ has precisely one occurrence of $\vdash_x$, precisely one occurrence of $\dashv_x$, and the former is before (i.e., to the left of) the latter.

▶ **Example 2.** The ref-word $\mathbf{r}_1 := \vdash_x \mathtt{aa} \dashv_y \vdash_x \mathtt{ab} \dashv_y$ is not valid for $\{x, y\}$ whereas the ref-words $\mathbf{r}_2 := \vdash_x \mathtt{aa} \dashv_x \vdash_y \mathtt{ab} \dashv_y$ and $\mathbf{r}_3 := \vdash_y \mathtt{a} \dashv_y \vdash_x \mathtt{a} \dashv_x \mathtt{ab}$ are valid for $\{x, y\}$. ⌟

To connect ref-words to terminal strings and later to spanners, we define a morphism $\mathsf{clr}\colon (\Sigma \cup \Gamma_X)^* \to \Sigma^*$ by $\mathsf{clr}(\sigma) := \sigma$ for $\sigma \in \Sigma$, and $\mathsf{clr}(\tau) := \epsilon$ for $\tau \in \Gamma_X$. For $\mathbf{d} \in \Sigma^*$, let $\mathsf{Ref}(\mathbf{d})$ be the set of all valid ref-words $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ with $\mathsf{clr}(\mathbf{r}) = \mathbf{d}$. By definition, every $\mathbf{r} \in \mathsf{Ref}(\mathbf{d})$ has a unique factorization $\mathbf{r} = \mathbf{r}'_x \cdot \vdash_x \cdot \mathbf{r}_x \cdot \dashv_x \cdot \mathbf{r}''_x$ for each $x \in X$. With these factorizations, we interpret $\mathbf{r}$ as a $(X, \mathbf{d})$-mapping $\mu^{\mathbf{r}}$ by defining $\mu^{\mathbf{r}}(x) := [i, j\rangle$, where $i := |\mathsf{clr}(\mathbf{r}'_x)| + 1$ and $j := i + |\mathsf{clr}(\mathbf{r}_x)|$. An alternative way of understanding $\mu^{\mathbf{r}} = [i, j\rangle$ is that i is chosen such that $\vdash_x$ occurs between the positions in $\mathbf{r}$ that are mapped to σ_{i-1} and σ_i, and $\dashv_x$ occurs between the positions that are mapped to σ_{j-1} and σ_j (assuming that $\mathbf{d} = \sigma_1 \cdots \sigma_{|\mathbf{d}|}$, and slightly abusing the notation to avoid a special distinction for the non-existing positions σ_0 and $\sigma_{|\mathbf{d}|+1}$).

▶ **Example 3.** Let $\mathbf{d} = \mathtt{aaab}$. The ref-word $\mathbf{r}_2$ from Example 2 is interpreted as the $(\{x, y\}, \mathbf{d})$-mapping $\mu^{\mathbf{r}_2}$ defined by $\mu^{\mathbf{r}_2}(x) := [1, 3\rangle$ and $\mu^{\mathbf{r}_2}(y) := [3, 5\rangle$. ⌟

Extraction grammars define ref-languages which are sets of ref-words. The ref-language $\mathcal{R}(G)$ of an extraction grammar $G := (X, V, \Sigma, P, S)$ is defined by $\mathcal{R}(G) := \{\mathbf{r} \in (\Sigma \cup \Gamma_X)^* \mid S \Rightarrow^* \mathbf{r}\}$. Note that we use $\mathcal{R}(G)$ instead of $\mathcal{L}(G)$ being used for standard grammars, to emphasize that the produced language is a ref-language. (We also use $\mathcal{L}(G)$ when G is a standard grammar.) To illustrate the definition let us consider the following example.

▶ **Example 4.** Both ref-words $\mathbf{r}_1$ and $\mathbf{r}_2$ from Example 2 are in $\mathcal{R}(\textsc{disjEqLen})$ where $\textsc{disjEqLen}$ is the grammar described in Example 1. Producing both $\mathbf{r}_1$ and $\mathbf{r}_2$ starts similarly with the sequence: $S \Rightarrow B \vdash_x A \dashv_y B \Rightarrow^2 \vdash_x A \dashv_y \Rightarrow \vdash_x \mathtt{a}A\mathtt{b} \dashv_y \Rightarrow \vdash_x \mathtt{aa}A\mathtt{ab} \dashv_y$. The derivation of $\mathbf{r}_1$ continues with $\Rightarrow \vdash_x \mathtt{aa} \dashv_y B \vdash_x \mathtt{ab} \dashv_y \Rightarrow \vdash_x \mathtt{aa} \dashv_y \vdash_x \mathtt{ab} \dashv_y$ whereas that of $\mathbf{r}_2$ continues with $\Rightarrow \vdash_x \mathtt{aa} \dashv_x B \vdash_y \mathtt{ab} \dashv_y \Rightarrow \vdash_x \mathtt{aa} \dashv_x \vdash_y \mathtt{ab} \dashv_y$. ⌟

We denote by $\mathsf{Ref}(G)$ the set of all ref-words in $\mathcal{R}(G)$ that are valid for X. Finally, we define the set $\mathsf{Ref}(G, \mathbf{d})$ of ref-words in $\mathsf{Ref}(G)$ that clr maps to $\mathbf{d}$. That is, $\mathsf{Ref}(G, \mathbf{d}) := \mathsf{Ref}(G) \cap \mathsf{Ref}(\mathbf{d})$. The result of evaluating the spanner $[\![G]\!]$ on a document $\mathbf{d}$ is then defined as

$$[\![G]\!](\mathbf{d}) := \{\mu^{\mathbf{r}} \mid \mathbf{r} \in \mathsf{Ref}(G, \mathbf{d})\}.$$

▶ **Example 5.** Let us consider the document $\mathbf{d} := \texttt{aaba}$. The grammar DISJEQLEN maps $\mathbf{d}$ into a set of $(\{x, y\}, \mathbf{d})$-mappings, amongst are $\mu^{\mathbf{r}_2}$ that is defined by $\mu^{\mathbf{r}_2}(x) := [1, 3\rangle$ and $\mu^{\mathbf{r}_2}(y) := [3, 5\rangle$ and $\mu^{\mathbf{r}_3}$ that is defined by $\mu^{\mathbf{r}_3}(x) := [2, 3\rangle$ and $\mu^{\mathbf{r}_3}(y) := [1, 2\rangle$. It can be shown that the grammar DISJEQLEN maps every document $\mathbf{d}$ into all possible $(\{x, y\}, \mathbf{d})$-mappings μ such that $\mu(x)$ and $\mu(y)$ are disjoint (i.e., do not overlap) and have the same length (i.e., $|\mathbf{d}_{\mu(x)}| = |\mathbf{d}_{\mu(y)}|$). ⌟

A spanner S is said to be *definable* by an extraction grammar G if $S(\mathbf{d}) = [\![G]\!](\mathbf{d})$ for every document $\mathbf{d}$.

▶ **Definition 6.** *A* context-free spanner *is a spanner definable by an extraction grammar.*

2.4 Extraction Pushdown Automata

An *extraction pushdown automaton*, or *extraction* PDA, is associated with a finite set $X \subseteq \mathsf{Vars}$ of variables and can be viewed as a standard pushdown automata over the extended alphabet $\Sigma \cup \Gamma_X$. Formally, an *extraction* PDA is a tuple $A := (X, Q, \Sigma, \Delta, \delta, q_0, Z, F)$ where X is a finite set of variables; Q is a finite set of states; Σ is the input alphabet; Δ is a finite set which is called *the stack alphabet*; δ is a mapping $Q \times \left(\Sigma \cup \{\epsilon\} \cup \Gamma_X\right) \times \Delta \to 2^{Q \times \Delta^*}$ which is called the *transition function*; $q_0 \in Q$ is the *initial state*; $Z \in \Delta$ is the *initial stack symbol*; and $F \subseteq Q$ is the set of *accepting states*. Indeed, extraction PDAs run on ref-words (i.e., finite sequences over $\Sigma \cup \Gamma_X$), as opposed to classical PDAs whose input are words (i.e., finite sequences over Σ). Similarly to classical PDAs, the computation of extraction PDAs can be described using sequences of configurations: a *configuration* of A is a triple (q, w, γ) where q is the state, w is the remaining input, and γ is the stack content such that the top of the stack is the left end of γ and its bottom is the right end. We use the notation $\vdash^*$ similarly to how it is used in the context of PDAs [21] and define the ref-language $\mathcal{R}(A)$:

$$\mathcal{R}(A) := \{\mathbf{r} \in (\Sigma \cup \Gamma_X)^* \mid \exists \alpha \in \Delta^*, q_f \in F : (q_0, \mathbf{r}, Z) \vdash^* (q_f, \epsilon, \alpha)\}.$$

We denote the language of A by $\mathcal{R}(A)$ to emphasize that it is a ref-language, and denote by $\mathsf{Ref}(A)$ the set of all ref-words in $\mathcal{R}(A)$ that are valid for X. The result of evaluating the spanner $[\![A]\!]$ on a document $\mathbf{d}$ is then defined as

$$[\![A]\!](\mathbf{d}) := \{\mu^{\mathbf{r}} \mid \mathbf{r} \in \mathsf{Ref}(A) \cap \mathsf{Ref}(\mathbf{d})\}.$$

▶ **Example 7.** We define the extraction PDA that maps a document $\mathbf{d}$ into the set of $(\{x, y\}, \mathbf{d})$-mappings μ where $\mu(x)$ ends before $\mu(y)$ starts and their lengths are the same. The stack alphabet consists of the bottom symbol $\perp$ and C, and the transition function δ is described in Figure 4 where a transition from state q to state q' that is labeled with $\tau, A/\ \gamma$ denotes that the automaton moves from state q to state q' upon reading τ with A at the top of the stack, while replacing A with γ. We can extend the automaton in a symmetric way such that it will represent the same spanner as that represented by the grammar DISJEQLEN from Example 1. ⌟

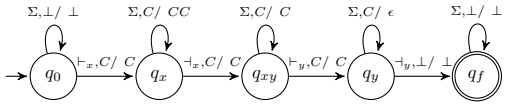


Figure 4 Transition function of Example 7.

$$S \to B \vdash_x A_1 \dashv_y B \mid B \vdash_y A_2 \dashv_x B$$
$$A_i \to \mathtt{a}A_i\mathtt{a} \mid \mathtt{a}A_i\mathtt{b} \mid \mathtt{b}A_i\mathtt{b} \mid \mathtt{b}A_i\mathtt{a},\ i = 1, 2$$
$$A_1 \to \dashv_x B \vdash_y , \quad A_2 \to \dashv_y B \vdash_x$$
$$B \to \epsilon \mid \mathtt{a}B \mid \mathtt{b}B$$

Figure 5 Productions of Example 9.

We say that a spanner S is *definable* by an extraction PDA A if for every document $\mathbf{d}$ it holds that $[\![A]\!](\mathbf{d}) = S(\mathbf{d})$. Treating the variable operations as terminal symbols enables us to use the equivalence of PDAs and context-free grammars and conclude the following straightforward observation.

▶ **Proposition 8.** *The class of spanners definable by extraction grammars is equal to the class of spanners definable by extraction PDAs.*

Thus, we have also an automata formalism for defining context-free spanners.

2.5 Functional Extraction Grammars

Freydenberger and Holldack [16] have presented the notion of *functionality* in the context of regular spanners. We now extend it to extraction grammars. The intuition is that interpreting an extraction grammar as a spanner disregards ref-words that are not valid. We call an extraction grammar G *functional* if every ref-word in $\mathcal{R}(G)$ is valid.

▶ **Example 9.** The grammar DISJEQLEN in our running example is not functional. Indeed, we saw in Example 4 that the ref-word $\mathbf{r}_1$, although it is not valid, is in $\mathcal{R}(\text{DISJEQLEN})$. We can, however, simply modify the grammar to obtain an equivalent functional one. Notice that the problem arises due to the production rules $S \to B \vdash_x A \dashv_y B$ and $S \to B \vdash_y A \dashv_x B$. For the non-terminal A we have $A \Rightarrow^* \mathbf{r}_1$ where $\mathbf{r}_1$ contains both $\dashv_x$ and $\vdash_y$, and we also have $A \Rightarrow^* \mathbf{r}_2$ where $\mathbf{r}_2$ contains both $\dashv_y$ and $\vdash_x$. To fix that, we can replace the non-terminal A with two non-terminals, namely A_1 and A_2, and change the production rules so that for every ref-word $\mathbf{r}$, if $A_1 \Rightarrow^* \mathbf{r}$ then $\mathbf{r}$ contains both $\dashv_x$ and $\vdash_y$, and if $A_2 \Rightarrow^* \mathbf{r}$ then $\mathbf{r}$ contains both $\dashv_y$ and $\vdash_x$. It can be shown that the grammar G whose production rules appear in Figure 5 is functional and that $[\![G]\!] = [\![\text{DISJEQLEN}]\!]$. ⌟

▶ **Proposition 10.** *Every extraction grammar G can be converted into an equivalent functional extraction grammar G' in $O(|G|^2 + 3^{2k}|G|)$ time where k is the number of variables G is associated with.*

Inspired by Chomsky's hierarchy, we say that an extraction grammar is in *Chomsky Normal Form (CNF)* if it is in CNF when viewed as a grammar over the extended alphabet $\Sigma \cup \Gamma_X$. We remark that, in Proposition 10, G' is in CNF.

2.6 Unambiguous Extraction Grammars

A grammar G is said to be unambiguous if every word it produces has a unique parse-tree. We extend this definition to extraction grammars. An extraction grammar G is said to be *unambiguous* if for every document $\mathbf{d}$ and every $(X, \mathbf{d})$-mapping $\mu \in [\![G]\!](\mathbf{d})$ it holds that there is a unique ref-word $\mathbf{r}$ for which $\mu^{\mathbf{r}} = \mu$ and this ref-word has a unique parse-tree. Unambiguous extraction grammars are less expressive than their ambiguous counterparts as the Boolean case shows – unambiguous context-free grammars are less expressive than ambiguous context-free grammars [22].

▶ **Example 11.** The extraction grammar given in Example 9 is not unambiguous since it produces the ref-words $\vdash_x \dashv_x \vdash_y \dashv_y$ and $\vdash_y \dashv_y \vdash_x \dashv_x$ that correspond to the same mapping. It can be shown that replacing the derivation $B \to \epsilon$ with $B \to \mathsf{a} \mid \mathsf{b}$ results in an unambiguous extraction grammar which is equivalent to DISJEQLEN on any document different than ϵ. (Note however that this does not imply that the ref-languages both grammars produce are equal.) ⌟

Our main enumeration algorithm for extraction grammars relies on unambiguity and the following observation.

▶ **Proposition 12.** *In Proposition 10, if G is unambiguous then so is G'.*

3 Expressive Power and Evaluation

In this section we compare the expressiveness of context-free spanners compared to other studied classes of spanners and discuss its evaluation shortly.

3.1 Regular Spanners

A *variable-set automaton* A (or *vset-automaton*, for short) is a tuple $A := (X, Q, q_0, q_f, \delta)$ where $X \subseteq \mathsf{Vars}$ is a finite set of variables also referred to as $\mathsf{Vars}(A)$, Q is the set of *states*, $q_0, q_f \in Q$ are the *initial* and the *final* states, respectively, and $\delta \colon Q \times (\Sigma \cup \{\epsilon\} \cup \Gamma_X) \to 2^Q$ is the *transition function*. To define the semantics of A, we interpret A as a non-deterministic finite state automaton over the alphabet $\Sigma \cup \Gamma_X$, and define $\mathcal{R}(A)$ as the set of all ref-words $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ such that some path from q_0 to q_f is labeled with $\mathbf{r}$. Like for regex formulas, we define $\mathsf{Ref}(A, \mathbf{d}) = \mathcal{R}(A) \cap \mathsf{Ref}(\mathbf{d})$ and finally we define for every document $\mathbf{d} \in \Sigma^*$: $[\![A]\!](\mathbf{d}) := \{\mu^{\mathbf{r}} \mid \mathbf{r} \in \mathsf{Ref}(A, \mathbf{d})\}$. The class of *regular spanners* equals the class of spanners that are expressible as a vset-automaton [11].

Inspired by Chomsky's hierarchy, we say that an extraction grammar G is *regular* if its productions are of the form $A \to \sigma B$ and $A \to \sigma$ where A, B are non-terminals and $\sigma \in (\Sigma \cup \Gamma_X)$. We then have the following equivalence that is strongly based on the equivalence of regular grammars and finite state automata.

▶ **Proposition 13.** *The class of spanners definable by regular extraction grammars is equal to the class of regular spanners.*

3.2 (Generalized) Core Spanners

An alternative way to define regular spanners is based on the notion of regex formulas: Formally, a *regex formula* is defined recursively by $\alpha := \emptyset \mid \epsilon \mid \sigma \mid \alpha \vee \alpha \mid \alpha \cdot \alpha \mid \alpha^* \mid \vdash_x \alpha \dashv_x$ where $\sigma \in \Sigma$ and $x \in \mathsf{Vars}$. We denote the set of variables whose variable operations occur in α by $\mathsf{Vars}(\alpha)$, and interpret each regex formula α as a generator of a ref-word language $\mathcal{R}(\alpha)$ over the extended alphabet $\Sigma \cup \Gamma_{\mathsf{Vars}(\alpha)}$. For every document $\mathbf{d} \in \Sigma^*$, we define $\mathsf{Ref}(\alpha, \mathbf{d}) = \mathcal{R}(\alpha) \cap \mathsf{Ref}(\mathbf{d})$, and the spanner $[\![\alpha]\!]$ by $[\![\alpha]\!](\mathbf{d}) := \{\mu^{\mathbf{r}} \mid \mathbf{r} \in \mathsf{Ref}(\alpha, \mathbf{d})\}$. The class of regular spanners is then defined as the closure of regex formulas under the relational algebra operators: union, projection and natural join. (See full definitions in [11].)

In their efforts to capture the core of AQL which is IBM's SystemT query language, Fagin et al. [11] have presented the class of core spanners which is the closure of regex formulas under the positive operators, i.e., union, natural join and projection, along with the string equality selection that is defined as follows Let S be a spanner and let $x, y \in \mathsf{Vars}(S)$, the *string equality selection* $\zeta^=_{x,y} S$ is defined by $\mathsf{Vars}(\zeta^=_{x,y} S) = \mathsf{Vars}(S)$ and, for all $\mathbf{d} \in \Sigma^*$,

$\zeta^{=}_{x,y}S(\mathbf{d})$ is the set of all $\mu \in S(\mathbf{d})$ where $\mathbf{d}_{\mu(x)} = \mathbf{d}_{\mu(y)}$. Note that unlike the join operator that joins mappings that have identical spans in their shared variables, the selection operator compares the substrings of $\mathbf{d}$ that are described by the spans, and does not distinguish between different spans that span the same substrings.

The class of *generalized core spanners* is obtained by adding the difference operator. That is, it is defined as the closure of regex formulas under union, natural join, projection, string equality, and difference. We say that two classes $\mathcal{S}, \mathcal{S}'$ of spanners are *incomparable* if both $\mathcal{S} \setminus \mathcal{S}'$ and $\mathcal{S}' \setminus \mathcal{S}$ are not empty.

▶ **Proposition 14.** *The classes of core spanners and generalized core spanners are each incomparable with the class of context-free spanners.*

We conclude the discussion by a straightforward result on closure properties.

▶ **Proposition 15.** *The class of context-free spanners is closed under union and projection, and not closed under natural join and difference.*

3.3 Evaluating Context-Free Spanners

The *evaluation* problem of extraction grammars is that of computing $[\![G]\!](\mathbf{d})$ where $\mathbf{d}$ is a document and G is an extraction grammar. Our first observation is the following.

▶ **Proposition 16.** *For every extraction grammar G and every document $\mathbf{d}$ it holds that $[\![G]\!](\mathbf{d})$ can be computed in $O(|G|^2 + |\mathbf{d}|^{2k+3}\, k^3\, |G|)$ time where k is the number of variables G is associated with.*

The proof of this proposition is obtained by iterating through all valid ref-words and using the Cocke-Younger-Kasami (CYK) parsing algorithm [22] to check whether the current valid ref-word is produced by G. We can, alternatively, use Valiant's parser [40] and obtain $O(|G|^2 + |\mathbf{d}|^{2k+\omega}\, k^\omega\, |G|)$ where $\omega < 2.373$ is the matrix multiplication exponent [42].

While the evaluation can be done in polynomial time in data complexity (where G is regarded as fixed and $\mathbf{d}$ as input), the output size might be quite big. To be more precise, for an extraction grammar G associated with k variables, the output might consist of up to $|\mathbf{d}|^{2k}$ mappings. Instead of outputting these mappings altogether, we can output them sequentially (without repetitions) after some preprocessing.

Our main enumeration result is the following.

▶ **Theorem 17.** *For every unambiguous extraction grammar G and every document $\mathbf{d}$ there is an algorithm that outputs the mappings in $[\![G]\!](\mathbf{d})$ with delay $O(k)$ after $O(|\mathbf{d}|^5|G|^2 3^{4k})$ preprocessing where k is the number of variables G is associated with.*

Our algorithm consists of two main stages: preprocessing and enumeration. In the preprocessing stage, we manipulate the extraction grammar and do some precomputations which are later exploited in the enumeration stage in which we output the results sequentially. We remark that unambiguity is crucial for the enumeration stage as it allows to output the mappings without repetition.

Through the lens of data complexity, our enumeration algorithm outputs the results with constant delay after quintic preprocessing. That should be contrasted with regular spanners for which there exists a constant delay enumeration algorithm whose preprocessing is linear [2, 13]. In the following sections, we present the enumeration algorithm and discuss its correctness but before we deal with the special case $\mathbf{d} := \epsilon$. In this case, $[\![G]\!](\mathbf{d})$ is either empty or contains exactly one mapping (since, by definition, the document ϵ has exactly

one span, namely $[1, 1\rangle$). Notice that $[\![G]\!](\mathbf{d})$ is empty if and only if G does not produce a ref-word that consists only of variable operations. To check this, it suffices to change the production rules of G by replacing every occurrence of $\tau \in \Gamma_X$ with ϵ, and checking whether the new grammar produces ϵ. This can be done in linear time [21], which completes the proof of this case. From now on it is assumed that $\mathbf{d} \neq \epsilon$.

4 Preprocessing of the Enumeration Algorithm

Due to Propositions 10 and 12, we can assume that our unambiguous extraction grammar is functional and in CNF. As this conversion requires $O(3^{2k}|G|^2)$, it can be counted as part of our preprocessing.

The preprocessing stage consists of two steps: in the first we adjust the extraction grammar to a given document and add subscripts to non-terminals to track this connection, and in the second we use superscripts to capture extra information regarding the variable operations.

4.1 Adjusting the Extraction Grammar to $\mathbf{d}$

Let $G := (X, V, \Sigma, P, S)$ be an extraction grammar in CNF, and let $\mathbf{d} := \sigma_1 \cdots, \sigma_n, n \geq 1$ be a document. The goal of this step is to restrict G so that it will produce only the ref-words which clr maps to $\mathbf{d}$. To this end, we define the grammar $G_\mathbf{d}$ that is associated with the same set X of variables as G, and is defined as follows:

- The non-terminals are $\{A_{i,j} \mid A \in V, 1 \leq i \leq j \leq n\} \cup \{A_\epsilon \mid A \in V\}$,
- the terminals are Σ,
- the initial non-terminal is $S_{1,n}$, and
- the production rules are defined as follows:
  - $A_{i,i} \to \sigma_i$ for any $A \to \sigma_i \in P$,
  - $A_\epsilon \to \sigma$ for any $A \to \sigma \in P$ with $\sigma \in \Gamma_X$,
  - $A_\epsilon \to B_\epsilon C_\epsilon$ for any $A \to BC \in P$,
  - $A_{i,j} \to B_{i,j} C_\epsilon$ for any $1 \leq i \leq j \leq n$ and any $A \to BC \in P$,
  - $A_{i,j} \to B_\epsilon C_{i,j}$ for any $1 \leq i \leq j \leq n$ and any $A \to BC \in P$,
  - $A_{i,j} \to B_{i,i'} C_{i'+1,j}$ for any $1 \leq i \leq i' < j \leq n$ and $A \to BC \in P$.

We eliminate useless non-terminals from $G_\mathbf{d}$ and by a slight abuse of notation refer to the result as $G_\mathbf{d}$ from now on. The intuition behind this construction is that if the subscript of a non-terminal is i, j then this non-terminal produces a ref-word that clr maps to $\sigma_i \cdots \sigma_j$, and if it is ϵ then it produces a ref-word that consists only of variable operations.

▶ **Example 18.** Figure 6 presents a possible parse-tree of a grammar $G_\mathbf{d}$. ⌟

We establish the following connection between G and $G_\mathbf{d}$.

▶ **Lemma 19.** *For every extraction grammar G in CNF, every document $\mathbf{d} := \sigma_1 \cdots \sigma_n$, every non-terminal A of G, and every ref-word $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ with $\mathsf{clr}(\mathbf{r}) = \sigma_i \cdots \sigma_j$ the following holds: $A \Rightarrow_G^* \mathbf{r}$ if and only if $A_{i,j} \Rightarrow_{G_\mathbf{d}}^* \mathbf{r}$*

This allows us to conclude the following straightforward corollary.

▶ **Corollary 20.** *For every extraction grammar G in CNF and for every document $\mathbf{d}$, it holds that $\mathsf{Ref}(G, \mathbf{d}) = \mathcal{L}(G_\mathbf{d})$.*

We note that adjusting our extraction grammar to $\mathbf{d}$ is somewhat similar to the CYK algorithm [22] and therefore it is valid on extraction grammars G in CNF. For a similar reason, we obtain the following complexity which is cubic in $|\mathbf{d}|$.

▶ **Proposition 21.** *For every extraction grammar G in CNF and for every document $\mathbf{d}$, it holds that $G_{\mathbf{d}}$ can be constructed in $O(|\mathbf{d}|^3|G|)$.*

Can the complexity of the adjustment be improved? We leave this as an open question. We note, however, that it might be possible to use similar ideas used by Earley's algorithm [10] to decrease the complexity of this step.

4.2 Constructing the Decorated Grammar

The goal of this step of the preprocessing is to encode the information on the produced variable operations within the terminals and non-terminals. We obtain from $G_{\mathbf{d}}$, constructed in the previous step, a new grammar, namely $\textsc{decorGrmr}(G_{\mathbf{d}})$, that produces *decorated words* over the alphabet $\{(\mathbf{x}, i, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \subseteq \Gamma_X, 1 \leq i \leq n\}$. A terminal $(\mathbf{x}, i, \mathbf{y})$ indicates that $\mathbf{x}$ and $\mathbf{y}$ are variable operations that occur right before and right after σ_i, respectively. (Notice that $\mathbf{x}, \mathbf{y}$ does not necessarily contain all of these variable operations as some of the variable operations that appear, e.g., after i, can be contained in $\mathbf{x}'$ in case $(\mathbf{x}', i+1, \mathbf{y}')$ is the terminal that appears right after $(\mathbf{x}, i, \mathbf{y})$.) This information is propagated also to the non-terminals such that a non-terminal with a superscript $\mathbf{x}, \mathbf{y}$ indicates that $\mathbf{x}$ and $\mathbf{y}$ are variable operations at the beginning and end, respectively, of the sub decorated word produced by this non-terminal. Non-terminals with subscript ϵ are those that produce sequences of variable operations.

To define $\textsc{decorGrmr}(G_{\mathbf{d}})$, we need G to be functional. The following key observation is used in the formal definition of $\textsc{decorGrmr}(G_{\mathbf{d}})$ and is based on the functionality of G.

▶ **Proposition 22.** *For every functional extraction grammar G and every non-terminal A of G there is a set $\mathbf{x}_A \subseteq \Gamma_X$ of variable operations such that for every ref-word $\mathbf{r}$ where $A \Rightarrow^* \mathbf{r}$ the variable operations that appear in $\mathbf{r}$ are exactly those in $\mathbf{x}_A$. Computing all sets $\mathbf{x}_A$ can be done in $O(|G|)$.*

In other words, for functional extraction grammars, the information on the variable operations is stored implicitly in the non-terminals. The grammar $\textsc{decorGrmr}(G_{\mathbf{d}})$ is defined in three steps as we now describe.

Step 1. We set the following production rules for all subsets $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w} \subseteq \Gamma_X$ that are pairwise disjoint:

- $A_{i,i}^{\emptyset,\emptyset} \to \sigma_i$ for every rule $A_{i,i} \to \sigma_i$ in $G_{\mathbf{d}}$,
- $A_\epsilon \to \epsilon$ for every rule $A_\epsilon \to \tau$ in $G_{\mathbf{d}}$ (with $\tau \in \Gamma_X$),
- $A_\epsilon \to B_\epsilon C_\epsilon$ for every rule $A_\epsilon \to B_\epsilon C_\epsilon$ in $G_{\mathbf{d}}$,
- $A_{i,j}^{\mathbf{x},\mathbf{y}\cup\mathbf{x}_C} \to B_{i,j}^{\mathbf{x},\mathbf{y}} C_\epsilon$ for every rule $A_{i,j} \to B_{i,j} C_\epsilon$ in $G_{\mathbf{d}}$ and $\mathbf{x} \cap \mathbf{x}_C = \mathbf{y} \cap \mathbf{x}_C = \emptyset$,
- $A_{i,j}^{\mathbf{x}\cup\mathbf{x}_B,\mathbf{y}} \to B_\epsilon C_{i,j}^{\mathbf{x},\mathbf{y}}$ for every rule $A_{i,j} \to B_\epsilon C_{i,j}$ in $G_{\mathbf{d}}$ and $\mathbf{x} \cap \mathbf{x}_B = \mathbf{y} \cap \mathbf{x}_B = \emptyset$,
- $A_{i,j}^{\mathbf{x},\mathbf{w}} \to B_{i,i'}^{\mathbf{x},\mathbf{y}} C_{i'+1,j}^{\mathbf{z},\mathbf{w}}$ for every rule $A_{i,j} \to B_{i,i'} C_{i'+1,j}$ in $G_{\mathbf{d}}$ and pairwise disjoint $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w}$,

with $\mathbf{x}_B$ and $\mathbf{x}_C$ defined as in Proposition 22.

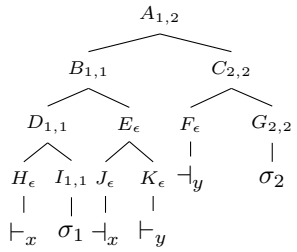


Figure 6 After the adjustment to $\mathbf{d}$.

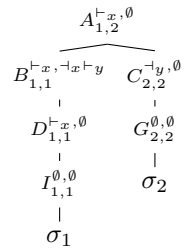


Figure 7 Before step 2 (*iii*).

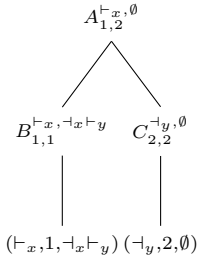


Figure 8 After step 3.

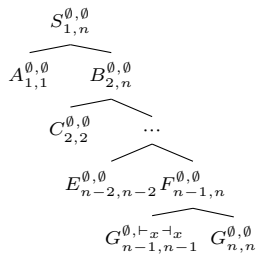


Figure 9 Non-stable non-terminals.

Step 2. We process the resulting grammar by three standard operations [22] in the following order: (*i*) we eliminate useless non-terminals (i.e., those that do not produce a terminal string or are not reachable from the initial non-terminal), (*ii*) we eliminate epsilon-productions, and (*iii*) we eliminate unit productions (i.e., rules of the form $A \to B$ were A, B are non-terminals). We elaborate on (*iii*) as it is important for the sequel. To eliminate unit productions we compute for each non-terminal the set of non-terminals that are reachable from it by unit productions only. That is, we say that a non-terminal B is *reachable* form non-terminal A if there is a sequence of unit productions of the form $A_1 \to A_2, \ldots, A_{n-1} \to A_n$ with $A_1 = A$ and $A_n = B$. We then replace every production $B \to \alpha$ which is not a unit production with $A \to \alpha$, and after that discard all unit productions.

Step 3. The last step of the construction is adding a fresh start symbol S and adding the production rules $S \to S_{1,n}^{\mathbf{x},\mathbf{y}}$ for every non-terminal of the form $S_{1,n}^{\mathbf{x},\mathbf{y}}$. We also replace each production of the form $A_{i,i}^{\mathbf{x},\mathbf{y}} \to \sigma_i$ with $A_{i,i}^{\mathbf{x},\mathbf{y}} \to (\mathbf{x}, i, \mathbf{y})$. This can be viewed as a "syntactic sugar" since it is only intended to help us formulate easily the connection between the grammar G and $\text{DECORGRMR}(G_{\mathbf{d}})$.

▶ **Example 23.** Figures 7 and 8 illustrate the different steps in the construction of the decorated grammar $\text{DECORGRMR}(G_{\mathbf{d}})$. For simplicity, we present the superscripts as pairs of sequences (each represent elements in the set) separated by commas ",". ⌟

Note that by a simple induction it can be shown that the resulting grammar does no longer contain non-terminals of the form A_ϵ. We denote the resulting grammar and its set of non-terminals by $\text{DECORGRMR}(G_{\mathbf{d}})$ and V^{DEC}, respectively.

The (X, d)-mapping μ^w that corresponds with $w := (\mathbf{x}_1, 1, \mathbf{y}_1) \cdots (\mathbf{x}_n, n, \mathbf{y}_n)$ (which is a decorated word produced by $\text{DECORGRMR}(G_{\mathbf{d}})$) is defined by $\mu^w(x) = [i, j\rangle$ where $\vdash_x \in \mathbf{x}_i \cup \mathbf{y}_{i-1}$ and $\dashv_x \in \mathbf{x}_j \cup \mathbf{y}_{j-1}$ with $\mathbf{y}_0 = \mathbf{x}_{n+1} = \emptyset$. We say that a decorated word w is *valid* if $\mu^w(x)$ is well-defined for every $x \in X$.

▶ **Proposition 24.** *For every functional extraction grammar G in CNF and for every document $\mathbf{d}$, if G is unambiguous then $\text{DECORGRMR}(G_{\mathbf{d}})$ is unambiguous.*

This allows us to establish the following connection between $\text{DECORGRMR}(G_{\mathbf{d}})$ and $[\![G]\!](\mathbf{d})$.

▶ **Lemma 25.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, every decorated word produced by $\text{DECORGRMR}(G_{\mathbf{d}})$ is valid and*

$$[\![G]\!](\mathbf{d}) = \{\mu^w \mid S \Rightarrow^*_{\text{DECORGRMR}(G_{\mathbf{d}})} w\}.$$

Finally, combining Proposition 24 and Lemma 25 leads to the following direct corollary.

▶ **Corollary 26.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, enumerating mappings in $[\![G]\!](\mathbf{d})$ can be done by enumerating parse-trees of decorated words in $\{w \mid S \Rightarrow^*_{\text{DECORGRMR}(G_\mathbf{d})} w\}$.*

To summarize the complexity of constructing $\text{DECORGRMR}(G_\mathbf{d})$ we have:

▶ **Proposition 27.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, $\text{DECORGRMR}(G_\mathbf{d})$ can be constructed in $O(|G_\mathbf{d}|\, 5^{2k}) = O(|\mathbf{d}|^3 |G|\, 5^{2k})$ where k is the number of variables associated with G.*

5 Enumeration Algorithm

Our enumeration algorithm builds recursively the parse-trees of the decorated grammar $\text{DECORGRMR}(G_\mathbf{d})$. Before presenting it, we discuss some of the main ideas that allow us to obtain a constant delay between every two consecutive outputs.

5.1 Stable non-terminals

The non-terminals of $\text{DECORGRMR}(G_\mathbf{d})$ are decorated with superscripts and subscripts that give extra information that can be exploited in the process of the derivation.

▶ **Example 28.** Figure 10 presents a partial parse-tree (without the leaves and the first production) for a decorated word in $\text{DECORGRMR}(G_\mathbf{d})$. Notice that the variable operations that appear in the subtrees rooted in the non-terminal $C_{1,3}^{\vdash_x \vdash_y, \dashv_y}$ are only those indicated in its superscript. That is, there are no variable operations that occur between positions $1, 2$, and no such between positions $2, 3$. ⌟

Motivated by this, we say that a non-terminal $A_{i,j}^{\mathbf{x},\mathbf{y}}$ of $\text{DECORGRMR}(G_\mathbf{d})$ is *stable* if $\mathbf{x}_A = \mathbf{x} \cup \mathbf{y}$.

▶ **Lemma 29.** *For every functional extraction grammar G in CNF and for every document $\mathbf{d}$, the set of stable non-terminals of $\text{DECORGRMR}(G_\mathbf{d})$ is computable in $O(|G_\mathbf{d}| 5^{2k})$ where k is the number of variables G is associated with.*

Therefore, while constructing the parse-trees of $\text{DECORGRMR}(G_\mathbf{d})$ whenever we reach a stable non-terminal we can stop since its subtree does not affect the mapping.

5.2 The Jump Function

If G is associated with k variables, there are exactly $2k$ variable operations in each ref-word produced by G. Hence, we can bound the number of non-stable non-terminals in a parse-tree of $\text{DECORGRMR}(G_\mathbf{d})$. Nevertheless, the depth of a non-stable non-terminal can be linear in $|\mathbf{d}|$ as the following example suggests.

▶ **Example 30.** Consider the non-stable non-terminal $F_{n-1,n}^{\emptyset,\emptyset}$ in the partial parse-tree in Figure 9 of the decorated word $(\emptyset, 1, \emptyset) \cdots (\emptyset, n-1, \vdash_x \dashv_x)(\emptyset, n, \emptyset)$. Observe that the depth of this non-terminal is linear in n. ⌟

Since we want the delay of our algorithm to be independent of $|\mathbf{d}|$, we skip parts of the parse-tree in which no variable operation occurs. This idea somewhat resembles an idea that was implemented by Amarilli et al. [2] in their constant delay enumeration algorithm for regular spanners represented as vset-automata. There, they defined a function that "jumps"

from one state to the other if the path from the former to the latter does not contain any variable operation. We extend this idea to extraction grammars by defining the notion of skippable productions. Intuitively, when we focus on a non-terminal in a parse-tree, the corresponding mapping is affected by either the left subtree of this non-terminal, or by its right subtree, or by the production applied on the non-terminal itself (or by any combination of the above). If the mapping is affected exclusively by the left (right, respectively) subtree then we can skip the production and move to check the left (right, respectively) subtree, and do so recursively until we reach a production for which this is no longer the case.

Formally, a *skippable* production rule is of the form $A^{\mathbf{x},\mathbf{y}}_{i,j} \to B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{z}',\mathbf{y}}_{i'+1,j}$ where (a) $A^{\mathbf{x},\mathbf{y}}_{i,j}$ is non-stable, (b) $\mathbf{z} = \mathbf{z}' = \emptyset$, and (c) exactly one of $B^{\mathbf{x},\mathbf{z}}_{i,i'}, C^{\mathbf{z}',\mathbf{y}}_{i'+1,j}$ is stable. Intuitively, (a) assures that the parse-tree rooted in $A^{\mathbf{x},\mathbf{y}}_{i,j}$ affects the mapping, (b) assures that the production applied on $A^{\mathbf{x},\mathbf{y}}_{i,j}$ does not affect the mapping and (c) assures that exactly one subtree of $A^{\mathbf{x},\mathbf{y}}_{i,j}$ (either the one rooted at $B^{\mathbf{x},\mathbf{z}}_{i,i'}$ if $C^{\mathbf{z}',\mathbf{y}}_{i'+1,j}$ is stable, or the one rooted at $C^{\mathbf{z}',\mathbf{y}}_{i'+1,j}$ if $B^{\mathbf{x},\mathbf{z}}_{i,i'}$ is stable) affects the mapping. We then say that a skippable production rule ρ *follows* a skippable production rule ρ' if the non-stable non-terminal in the right-hand side of ρ' is the non-terminal in the left-hand side of ρ. The function $\textsc{jump}\colon V^{\text{DEC}} \to 2^{V^{\text{DEC}}}$ is defined by $B \in \textsc{jump}(A^{\mathbf{x},\mathbf{y}}_{i,j})$ if there is a sequence of skippable production rules $\rho_1, \cdots, \rho_m$ such that:

- ρ_ι follows $\rho_{\iota-1}$ for every ι,
- the left-hand side of ρ_1 is $A^{\mathbf{x},\mathbf{y}}_{i,j}$,
- the non-stable non-terminal in the right-hand side of ρ_m is B,
- there is a production rule that is not skippable whose left-hand side is B.

▶ **Example 31.** In the decorated grammar whose (one of its) parse-tree appears in Figure 9 it holds that $F^{\emptyset,\emptyset}_{n-1,n} \in \textsc{jump}(S^{\emptyset,\emptyset}_{1,n})$. ⌟

The acyclic nature of the decorated grammar (that is, the fact that a non-terminal cannot be produced from itself) enables us to obtain the following upper bound for the computation of the JUMP function.

▶ **Lemma 32.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, the* JUMP *function is computable in $O(|\mathbf{d}|^5 3^{4k} |G|^2)$ where k is the number of variables G is associated with.*

Lemmas 29 and 32 imply that we can find the non-stable non-terminals as well as compute the JUMP function as part of the quintic preprocessing. It is important to note that if we can reduce the complexity of computing the JUMP function to cubic then we can reduce the whole preprocessing time to cubic.

5.3 The Algorithm

Our main enumeration algorithm is presented in Algorithm 1 and outputs $(X, \mathbf{d})$-mappings μ represented as sets of pairs $(\vdash_x, i), (\dashv_x, j)$ whenever $\mu(x) = [i, j\rangle$. The procedure APPLYPROD is called with a non-terminal $A^{\mathbf{x},\mathbf{y}}_{i,j}$ that is (a) non-stable and (b) appears at the left-hand side of at least one rule that is not skippable. The iterator APPLYPROD outputs with constant delay all those pairs (β, map) for which there exists a not skippable rule of the form $A^{\mathbf{x},\mathbf{y}}_{i,j} \to B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{z}',\mathbf{y}}_{i'+1,j}$ such that the following hold:

- $\mathsf{map} = \{(\tau, i'+1) \mid \tau \in \mathbf{z} \cup \mathbf{z}'\}$, and
- β is the concatenation of the non-stable terminals amongst $B^{\mathbf{x},\mathbf{z}}_{i,i'}$ and $C^{\mathbf{z}',\mathbf{y}}_{i'+1,j}$.

Notice that since $A^{\mathbf{x},\mathbf{y}}_{i,j}$ is non-stable and since $A^{\mathbf{x},\mathbf{y}}_{i,j} \to B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{z}',\mathbf{y}}_{i'+1,j}$ is not skippable, it holds that either $\mathbf{z} \neq \emptyset$ or $\mathbf{z}' \neq \emptyset$ (or both). Thus, the returned map is not empty which implies

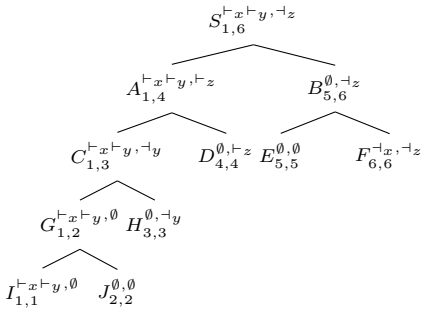


Figure 10 DECORGRMR($G_{\mathbf{d}}$) Parse tree.

Algorithm 1 Main enumeration algorithm.

procedure ENUMERATE(α, map)
if $\alpha = \epsilon$ **then**
  output map
denote α *by* $A \cdot \alpha'$;
foreach $B \in$ JUMP(A) **do**
  foreach $(\beta, \mathsf{map}') \in$ APPLYPROD(B) **do**
    ENUMERATE($\beta \cdot \alpha', \mathsf{map} \cup \mathsf{map}'$);

that every call to this procedure adds information on the mapping, and thus the number of calls is bounded. Notice also that β is the concatenation of the non-terminals among $B^{\mathbf{x},\mathbf{z}}_{i,i'}$ and $C^{\mathbf{z}',\mathbf{y}}_{i'+1,j}$ that affect the mapping.

▶ **Example 33.** The procedure APPLYPROD applied on $S^{\vdash_x \vdash_y, \dashv_z}_{1,6}$ from Figure 10 adds the pair $(\vdash_z, 5)$ to map; When applied on $A^{\vdash_x \vdash_y, \vdash_z}_{1,4}$, it adds the pair $(\dashv_y, 4)$ to map; When applied on $B^{\emptyset, \dashv_z}_{5,6}$, it adds the pair $(\dashv_x, 6)$ to map. ⌟

The recursive procedure ENUMERATE outputs the mapping as a set of pairs of the form (γ, i) with $\gamma \in \Gamma_X$ a variable operation and $1 \leq i \leq n$. The main enumeration algorithm calls the recursive procedure ENUMERATE with pairs $(S^{\mathbf{x},\mathbf{y}}_{1,n}, \mathsf{map})$ where $S^{\mathbf{x},\mathbf{y}}_{1,n}$ is a non-terminal in DECORGRMR($G_{\mathbf{d}}$), and map is the set containing pairs $(\tau, 1)$ for any $\tau \in \mathbf{x}$, and $(\tau, n+1)$ for any $\tau \in \mathbf{y}$. The recursive procedure ENUMERATE gets a pair (α, map) as input where α is a (possibly empty) sequence of non-stable non-terminals and map is a set of pairs of the above form. It recursively constructs an output mapping by applying derivations on the non-stable non-terminals (by calling APPLYPROD) while skipping the skippable productions (by using JUMP). We assume that ENUMERATE has $O(1)$ access to everything computed in the preprocessing stage, that is, the grammar DECORGRMR($G_{\mathbf{d}}$), the JUMP function, and the sets of stable and non-stable non-terminals.

▶ **Theorem 34.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, the main enumerating algorithm described above enumerates the mappings in $[\![G]\!](\mathbf{d})$ (without repetitions) with delay of $O(k)$ between each two consecutive mappings where k is the number of variables G is associated with.*

Had G been ambiguous, the complexity guarantees on the delay would not have held.

Finally, we remark that the proof of Theorem 17 follows from Corollary 25, Proposition 27, Lemma 29, Lemma 32, and Theorem 34.

6 Conclusion

In this paper we propose a new grammar-based language for document spanners, namely extraction grammars. We compare the expressiveness of context-free spanners with previously studied classes of spanners and present a pushdown model for these spanners. We present an enumeration algorithm for unambiguous grammars that outputs results with a constant delay after quintic preprocessing in data complexity. We conclude by suggesting several future research directions.

To reach a full understanding of the expressiveness of context-free spanners, one should characterize the string relations that can be expressed with context-free spanners. This can be done by understanding the expressiveness of context-free grammars enriched with string equality selection. We note that there are some similarities between recursive Datalog over regex formulas [34] and extraction grammars. Yet, with the former we reach the full expressiveness of polynomial time spanners (data complexity) whereas with the latter we cannot express string equality. Understanding the connection between these two formalisms better can be a step in understanding the expressive power of extraction grammars.

Regarding our enumeration complexity, it might be possible to decrease the preprocessing complexity by using other techniques to compute the jump function. Another direction is to find restricted classes of extraction grammars that are more expressive than regular spanners yet allow linear time preprocessing (similarly to [2]).

It can be interesting to examine more carefully whether the techniques used here for enumerating the derivations can be applied also for enumerating queries on trees, or enumerating queries beyond MSO on strings. This connects to a recent line of work on efficient enumeration algorithms for monadic-second-order queries on trees [3]. Can our techniques be used to obtain efficient evaluation for more expressive queries?

References

1 Jitendra Ajmera, Hyung-Il Ahn, Meena Nagarajan, Ashish Verma, Danish Contractor, Stephen Dill, and Matthew Denesuk. A CRM system for social media: challenges and experiences. In *WWW*, pages 49–58. ACM, 2013.

2 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, pages 22:1–22:19, 2019.

3 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *PODS*, pages 89–103, 2019.

4 Edward Benson, Aria Haghighi, and Regina Barzilay. Event discovery in social media feeds. In *ACL*, pages 389–398. The Association for Computer Linguistics, 2011.

5 J. Richard Büchi and Lawrence H. Landweber. Definability in the monadic second-order theory of successor. *J. Symb. Log.*, 34(2):166–170, 1969.

6 Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. SystemT: An algebraic approach to declarative information extraction. In *ACL*, pages 128–137, 2010.

7 Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-correctness in information extraction. In *PODS*, pages 149–163, 2019.

8 Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. In *ICDT*, volume 155 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

9 Pál Dömösi. Unusual algorithms for lexicographical enumeration. *Acta Cybernetica*, 14(3):461–468, 2000.

10 Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

11 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015.

12 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Declarative cleaning of inconsistencies in information extraction. *ACM Trans. Database Syst.*, 41(1):6:1–6:44, 2016.

13 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, pages 165–177. ACM, 2018.

14 Dominik D. Freydenberger. A logic for document spanners. In *ICDT*, volume 68 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

15 Dominik D. Freydenberger. A logic for document spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019.

16 Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. In *ICDT*, volume 48 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

17 Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. *Theory Comput. Syst.*, 62(4):854–898, 2018.

18 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *PODS*, pages 137–149, 2018.

19 Dominik D. Freydenberger and Liat Peterfreund. Finite models and the theory of concatenation. *CoRR*, abs/1912.06110, 2019. URL: `http://arxiv.org/abs/1912.06110`.

20 Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In *ICDT*, volume 155, pages 11:1–11:21, 2020.

21 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.

22 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007.

23 Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

24 Donald E. Knuth. Correction: Semantics of context-free languages. *Mathematical Systems Theory*, 5(1):95–96, 1971.

25 Clemens Lautemann, Thomas Schwentick, and Denis Thérien. Logics for context-free languages. In *CSL*, volume 933 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 1994.

26 Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. SVM based learning system for information extraction. In *Deterministic and Statistical Methods in Machine Learning, First International Workshop, Sheffield, UK, September 7-10, 2004, Revised Lectures*, pages 319–339, 2004.

27 Yunyao Li, Frederick Reiss, and Laura Chiticariu. SystemT: A declarative information extraction system. In *ACL*, pages 109–114. ACL, 2011.

28 Erkki Mäkinen. On lexicographic enumeration of regular and context-free languages. *Acta Cybernetica*, 13(1):55–61, 1997.

29 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. Document spanners for extracting incomplete information: Expressiveness and complexity. In *PODS*, pages 125–136, 2018.

30 Andrea Moro, Marco Tettamanti, Daniela Perani, Caterina Donati, Stefano F Cappa, and Ferruccio Fazio. Syntax and the brain: disentangling grammar by selective anomalies. *Neuroimage*, 13(1):110–118, 2001.

31 Takashi Nagashima. A formal deductive system for CFG. *Hitotsubashi journal of arts and sciences*, 28(1):39–43, 1987.

32 Yoav Nahshon, Liat Peterfreund, and Stijn Vansummeren. Incorporating information extraction in the relational database model. In *WebDB*, page 6. ACM, 2016.

33 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *PODS*, pages 320–334. ACM, 2019.

34 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *ICDT*, volume 127 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

35 Christopher De Sa, Alexander Ratner, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. Deepdive: Declarative knowledge base construction. *SIGMOD Record*, 45(1):60–67, 2016.

36 Markus L. Schmid. Characterising REGEX languages by regular languages equipped with factor-referencing. *Inf. Comput.*, 249:1–17, 2016.

37 Rania A Abul Seoud, Abou-Bakr M Youssef, and Yasser M Kadah. Extraction of protein interaction information from unstructured text using a link grammar parser. In *2007 International Conference on Computer Engineering & Systems*, pages 70–75. IEEE, 2007.

38 Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative information extraction using Datalog with embedded extraction predicates. In *VLDB*, pages 1033–1044, 2007.

39 Jason M Smith and David Stotts. SPQR: Flexible automated design pattern extraction from source code. In *18th IEEE International Conference on Automated Software Engineering*, pages 215–224. IEEE, 2003.

40 Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

41 Huang Wen-Ji. Enumerating sentences of context free language based on first one in order. *Journal of Computer Research and Development*, 41(1):9–14, 2004.

42 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898. ACM, 2012.

43 Hua Xu, Shane P. Stenner, Son Doan, Kevin B. Johnson, Lemuel R. Waitman, and Joshua C. Denny. MedEx: a medication information extraction system for clinical narratives. *JAMIA*, 17(1):19–24, 2010. doi:10.1197/jamia.M3378.

44 Akane Yakushiji, Yuka Tateisi, Yusuke Miyao, and Jun-ichi Tsujii. Event extraction from biomedical papers using a full parser. In *Biocomputing 2001*, pages 408–419. World Scientific, 2000.

45 Huaiyu Zhu, Sriram Raghavan, Shivakumar Vaithyanathan, and Alexander Löser. Navigating the intranet with high precision. In *WWW*, pages 491–500. ACM, 2007.

Input–Output Disjointness for Forward Expressions in the Logic of Information Flows

Heba Aamer
Hasselt University, Belgium

Jan Van den Bussche
Hasselt University, Belgium

Abstract

Last year we introduced the logic FLIF (forward logic of information flows) as a declarative language for specifying complex compositions of information sources with limited access patterns. The key insight of this approach is to view a system of information sources as a graph, where the nodes are valuations of variables, so that accesses to information sources can be modeled as edges in the graph. This allows the use of XPath-like navigational graph query languages. Indeed, a well-behaved fragment of FLIF, called io-disjoint FLIF, was shown to be equivalent to the executable fragment of first-order logic. It remained open, however, how io-disjoint FLIF compares to general FLIF. In this paper we close this gap by showing that general FLIF expressions can always be put into io-disjoint form.

2012 ACM Subject Classification Software and its engineering → Semantics; Software and its engineering → Data flow languages; Theory of computation → Database query languages (principles)

Keywords and phrases Composition, expressive power, variable substitution

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.8

Funding This work was partially supported by Artificial Intelligence Research Flanders.
Heba Aamer: Supported by the Special Research Fund (BOF) (BOF19OWB16).
Jan Van den Bussche: Partially supported by the National Natural Science Foundations of China (61972455).

Acknowledgements Thanks to Eugenia Ternovska for introducing us to LIF and to Bart Bogaerts for initial discussions on the topic of this paper.

1 Introduction

FLIF (Forward Logic of Information Flows) [1] is an algebraic language for accessing atomic modules, and composing such modules into complex transition systems. Here, the term "atomic module" can be interpreted liberally: it can be a software library function, a Web service, a form on a website, or an information source.

Abstractly, an atomic module may be viewed as an n-ary relation where some of the arguments are designated as input arguments, with the remaining arguments providing output. For a simple example, a telephone directory may be viewed as a binary relation Dir(name; phone) with name as input argument and phone as output argument. For another example, the public bus company may provide its weekdays schedule as a relation Route(stop, interval; time, line, next, duration) that, given a bus stop and a time interval, outputs bus lines that stop there at a time within the interval, together with the duration to the next stop. Note how we use a semicolon to separate the input arguments from the output arguments.

In database research, such relations are known as information sources with limited access patterns, and the querying of such sources has received considerable attention. We refer to the research monograph by Benedikt et al. for more background [7]. An elegant syntactic fragment of first-order logic (FO) that takes into account the limitations imposed by the access patterns, was proposed by Nash and Ludäscher in the form of so-called "executable"

FO formulas [14]. Executable FO queries can be evaluated by a form of relational algebra expressions, called *plans*, in which database relations can only be accessed by joining them on their input attributes with a relation that is either given as input or has already been computed.

FLIF now offers an alternative language, which we think of as situated halfway between executable FO and plans. In FLIF we take a novel graph-based perspective to information sources with access patterns. The nodes of the graph are variable bindings; edges indicate accesses to the sources. For example, consider a source Friend(pname; fname) that outputs names of friends when given the name of a person as input. Then there is an edge labeled Friend from binding ν_1 to binding ν_2 if ν_2(fname) is a friend of ν_1(pname). Moreover, ν_2 should not differ from ν_1 in other variables (a principle of "inertia"). FLIF then is a simple XPath-like navigational query language over such graphs [15, 10, 5, 9, 16, 4].

For example, abbreviating Friend by F, the FLIF expression $F(x;y);F(y;z);F(z;u);(u = x)$ retrieves (in variable z) friends of friends of x who have x also as a direct friend. Here, the operator ; denotes composition and the subexpression $(u = x)$ serves as a test. Composition is a crucial operator by which (bounded-length) paths can be traced in the graph. FLIF also has union (to branch off), intersection (to merge branches) and difference (to exclude branches), and variable assignments.

Simple as FLIF may seem, it is at least as expressive as executable FO, as we showed together with Bogaerts, Surinx and Ternovska at ICDT 2020 [1]. Actually, executable FO was shown to be already subsumed by a well-behaved fragment of FLIF, formed by the expressions that are *io-disjoint*. IO-disjointness is a syntactic restriction which guarantees that, during the transitions involved in evaluating the expression, the values of input variables are never overwritten. IO-disjoint expressions can also be evaluated by a very transparent translation to relational algebra plans, in which all joins are natural joins, and attribute renaming is not needed. (The example expression above is io-disjoint.)

Conversely, in our previous work we also gave a translation from io-disjoint FLIF to executable FO. It has remained open, however, whether every FLIF expression can actually be put in an equivalent form that is io-disjoint. In the present paper we answer this question affirmatively. Of course, we need to make precise for what kind of "equivalence" this can work, and it is part of our contribution to clarify this. Intuitively, we show that it is always possible to designate a fresh set of output variables disjoint from the set of input variables, in such a way that intermediate variables (used in subexpressions) do not interfere with either of the two sets. Proving this rigorously turned out to be a quite intricate task. Our result shows that io-disjoint FLIF is equally powerful as the full FLIF language. As a corollary, we obtain that full FLIF is not more powerful than executable FO.

This paper is organized as follows. Section 2 gives preliminaries. Section 3 gives examples of queries expressed in FLIF and other languages. Section 4 presents and discusses our main result. Section 5 formally proves the correctness of our method. Section 6 concludes.

2 Preliminaries

We begin by recalling from previous work the concepts and results needed for the present paper.

2.1 Syntax of FLIF

A *schema* consists of a set of *module names*, which are seen as relation names. Accordingly, the schema assigns to each module name M an *arity* $\mathrm{ar}(M)$, as well as an *input arity* $\mathrm{iar}(M)$ with $\mathrm{iar}(M) \leq \mathrm{ar}(M)$. The idea is that the first $\mathrm{iar}(M)$ arguments are the input arguments;

the remaining arguments are the output arguments. We use $\mathrm{oar}(M)$ (output arity) for $\mathrm{ar}(M) - \mathrm{iar}(M)$.

Assume countably infinite universes **dom** of data values, and $\mathbb{V}$ of variables. The expressions α of FLIF over a schema $\mathcal{S}$ are generated by the following grammar:

$$\alpha ::= M(\bar{x};\bar{y}) \mid (x := y) \mid (x := c) \mid (x = y) \mid (x = c) \mid \alpha\,;\alpha \mid \alpha \cup \alpha \mid \alpha \cap \alpha \mid \alpha - \alpha$$

Here, M is a module name from $\mathcal{S}$; $\bar{x}$ and $\bar{y}$ are tuples of variables of lengths $\mathrm{iar}(M)$ and $\mathrm{oar}(M)$ respectively; x and y are variables; and c is a constant from **dom**. An expression of the form $M(\bar{x};\bar{y})$ is called a *module access*; of the form $x := y$ or $x := c$ a *variable assignment*; and $x = y$ or $x = c$ an *equality test*.

▶ Remark 1. In writing expressions, we omit parentheses around (sub)expressions involving composition since it is an associative operator. Also we give precedence to composition over the set operations.

2.2 Semantics of FLIF

Recall that a *valuation* is a mapping from $\mathbb{V}$ to **dom**. It is convenient to be able to apply valuations also to constants, agreeing that $\nu(c) = c$ for any valuation ν and any $c \in \mathbf{dom}$. We use $\mathcal{V}$ for the set of all valuations.

The semantics of FLIF expressions are defined in the context of interpretations. An *interpretation* D of a schema $\mathcal{S}$ assigns to each module name M an n-ary relation $D(M)$ on **dom**, with $n = \mathrm{ar}(M)$. Given an interpretation D, we define the semantics of an expression α on D, denoted by $[\![\alpha]\!]_D$, as a binary relation on $\mathcal{V}$, as follows.

For module access:

$$[\![M(\bar{x};\bar{y})]\!]_D = \{(\nu_1,\nu_2) \in \mathcal{V}\times\mathcal{V} \mid \nu_1(\bar{x})\cdot\nu_2(\bar{y}) \in D(M) \text{ and } \nu_2 \text{ agrees with } \nu_1 \text{ outside } \bar{y}\}$$

where $\cdot$ denotes concatenation.

For variable assignment, where t is a variable or a constant:

$$[\![x := t]\!]_D = \{(\nu_1,\nu_2) \in \mathcal{V}\times\mathcal{V} \mid \nu_2 = \nu_1[x := \nu_1(t)]\}$$

where, in general, for a valuation ν, a variable x, and a data value c, we use $\nu[x := c]$ for the valuation that is the same as ν except that x is mapped to c.

For equality test:

$$[\![x = t]\!]_D = \{(\nu,\nu) \mid \nu \in \mathcal{V} \text{ and } \nu(x) = \nu(t)\}.$$

Finally, the semantics of the operators ;, $\cup$, $\cap$ and $-$ are given by composition of binary relations, union, intersection and set difference, respectively.[1]

▶ Remark 2. For simplicity of exposition, we only include equality tests and simple assignments. The definitions and results of this paper can however be extended to include arithmetical comparisons and operations.

▶ **Example 3.** Suppose F is a binary relation of input arity one that is interpreted as a symmetric "friends" relation of a social network. Suppose the value of input variable x is some famous person, and we want to find persons z who are friend of a friend (say, y) of x.

[1] Recall that the composition $r\,;s$ of two binary relations equals the binary relation $\{(\nu_1,\nu_3) \mid \exists\nu_2 : (\nu_1,\nu_2) \in r \text{ and } (\nu_2,\nu_3) \in s\}$.

Table 1 Input and output variables of FLIF expressions [1]. In the case of $M(\bar{x};\bar{y})$, the set X is the set of variables in $\bar{x}$, and the set Y is the set of variables in $\bar{y}$. The symbol $\triangle$ denotes symmetric difference.

α	$I(\alpha)$	$O(\alpha)$
$M(\bar{x};\bar{y})$	X	Y
$x = y$	$\{x, y\}$	$\emptyset$
$x := y$	$\{y\}$	$\{x\}$
$x = c$	$\{x\}$	$\emptyset$
$x := c$	$\emptyset$	$\{x\}$
$\alpha_1 \,;\, \alpha_2$	$I(\alpha_1) \cup (I(\alpha_2) \setminus O(\alpha_1))$	$O(\alpha_1) \cup O(\alpha_2)$
$\alpha_1 \cup \alpha_2$	$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$	$O(\alpha_1) \cup O(\alpha_2)$
$\alpha_1 \cap \alpha_2$	$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$	$O(\alpha_1) \cap O(\alpha_2)$
$\alpha_1 - \alpha_2$	$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$	$O(\alpha_1)$

For this we can use the simple expression $F(x;y) \,;\, F(y;z)$. In order to find persons z with a pair (say, y_1 and y_2) of friends of x, we can write the expression

$$(F(x;y_1) \,;\, F(x;y_2) \,;\, F(y_1;z)) \cap (F(x;y_1) \,;\, F(x;y_2) \,;\, F(y_2;z)).$$

If we want to make sure that y_1 and y_2 are different, we can write $\alpha - (\alpha \,;\, (y_1 = y_2))$ where α is the previous expression. In view of Remark 2 above, we could easily extend FLIF with nonequalities and then write $\alpha \,;\, (y_1 \neq y_2)$ instead.

2.3 Input and output variables

Intuitively, an *output variable* of an expression α is any variable whose value may change in the course of evaluating α. The set of *input variables* may then be defined as the smallest set of variables whose values determine the values of the output variables. These semantic notions of input and output are undecidable, as they are related to satisfiability in the algebra of binary relations [3]. Hence, we use syntactical overapproximations [1]. Thus Table 1 defines, for each expression α, the sets $I(\alpha)$ and $O(\alpha)$ of input and output variables.

The following inertia [12, 11] and input determinacy properties formally confirm that Table 1 defines correct overapproximations, i.e., $I(\alpha)$ contains all semantic input variables, and $O(\alpha)$ contains all semantic output variables. Elsewhere [2] we have shown that slight variants of our definitions of I and O are optimal among all compositional definitions satisfying the below two properties.

▶ **Proposition 4** ([1]). *Let α be an FLIF expression and let D be an interpretation.*
Inertia: *For any $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$, we have that ν_2 agrees with ν_1 outside $O(\alpha)$.*
Input determinacy: *Let $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ and let ν_1' be a valuation that agrees with ν_1 on $I(\alpha)$. Then there exists a valuation ν_2' that agrees with ν_2 on $O(\alpha)$, such that $(\nu_1', \nu_2') \in [\![\alpha]\!]_D$.*

One can verify that every variable occurring in an expression according to the definitions in Table 1 is either an input variable, an output variable, or both. Since FLIF lacks an explicit quantification operator, we also refer to the variables occurring in an expression α as the "free variables", denoted by $\mathrm{var}(\alpha)$, so $\mathrm{var}(\alpha) = I(\alpha) \cup O(\alpha)$. The following property

formalizes, as expected, that the evaluation of an expression is oblivious to the values of the non-free variables; they can take any values. (Of course by inertia, these values will not change in the course of evaluating the expression.)

▶ **Proposition 5** (Free variable property [1]). *Let Y be the set of variables* not *in* $\mathrm{var}(\alpha)$*, let* $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$*, and let* $\nu : Y \to \mathbf{dom}$ *be arbitrary. Then also* $(\nu_1[\nu], \nu_2[\nu]) \in [\![\alpha]\!]_D$*.*

Here, we use $\nu_i[\nu]$ for ν_i updated with ν, or formally, the valuation $\nu_i|_{\mathrm{var}(\alpha)} \cup \nu$.

▶ **Example 6.** Let us denote the expression $R(x;y) \cup S(x;z)$ by α. The definitions in Table 1 yield that $O(\alpha) = \{y, z\}$ and $I(\alpha) = \{x, y, z\}$. Having y and z as input variables may at first sight seem counterintuitive. To see semantically why, say, y is an input variable for α, consider an interpretation D where S contains the pair $(1, 3)$. Consider the valuation $\nu_1 = \{(x, 1), (y, 2), (z, 0)\}$, and let $\nu_2 = \nu_1[z := 3]$. Clearly $(\nu_1, \nu_2) \in [\![S(x;z)]\!]_D \subseteq [\![\alpha]\!]_D$. However, if we change the value of y in ν_1, letting $\nu_1' = \nu_1[y := 4]$, then (ν_1', ν_2) neither belongs to $[\![S(x;z)]\!]_D$ nor to $[\![R(x;y)]\!]_D$ (due to inertia). Thus, input determinacy would be violated if y would not belong to $I(\alpha)$.

We mentioned that Table 1 overapproximates the undecidable semantic notions of inputs and outputs. The following example provides a simple illustration.

▶ **Example 7.** Let α be the expression $(x = y)\,;R(x;y)\,;(x = y)$. It is clear that α does not have any ouputs, however $O(\alpha) = \{y\}$ according to the definitions of Table 1.

2.4 Input–output disjointness

An expression α is called *io-disjoint* if $I(\beta)$ and $O(\beta)$ are disjoint, for every subexpression β of α (including α itself). While there is nothing really "wrong" with expressions that are not io-disjoint, the io-disjoint expressions enjoy a particularly transparent evaluation process in which input slots remain intact while output slots are being filled.

▶ **Example 8.** Continuing Example 3 (friends), the expression $F(x;x)$ is obviously not io-disjoint. Evaluating this expression will overwrite variable x with a friend of the person originally stored in x. In contrast, the example expressions given in Example 3 are all io-disjoint. ◀

Formally, we have the following useful property, which follows from Proposition 4:

▶ **Proposition 9** (Identity property). *Let α be an io-disjoint expression and let D be an interpretation. If* $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$*, then also* $(\nu_2, \nu_2) \in [\![\alpha]\!]_D$*.*

Intuitively, the identity property holds because, if in ν_1 the output slots would accidentally already hold a correct combination of output values, then there will exist an evaluation of α that merely confirms these values.

▶ **Example 10.** The identity property clearly need not hold for expressions that are not io-disjoint. For example, continuing the friends example, for the expression $F(x;x)$, a person need not be a friend of themselves.

3 FLIF and other languages

In this Section we compare and contrast FLIF with other formalisms considered for querying over limited access patterns, specifically, executable FO; plans; and executable Datalog. For executable FO and plans we use the syntax introduced in our ICDT 2020 paper [1]. The syntax for Datalog needs no introduction.

▶ **Example 11.** Recall the Friends relation from Example 3. We will consider the query from that example in Example 12; here, we first consider a simpler query. Given a person x, we want in variable z the friends of a friend of x that are not friend with x itself.

In FLIF, we can express this query using only the variables x and z with the following simple expression:

$$F(x;z)\,;F(z;z)-F(x;z)$$

The same query can be expressed with the following io-disjoint FLIF expression:

$$F(x;y)\,;(F(y;z)-F(x;z))$$

The algebraic plan equivalent to this expression is:

$$\pi_z((In\bowtie F(x;y)\bowtie F(y;z))-(In\bowtie F(x;y)\bowtie F(x;z)))$$

where In is a relation name over $\{x\}$ providing input values.

In executable FO, the query will be expressed by the formula:

$$\exists y\, F(x;y)\wedge F(y;z)\wedge\neg F(x;z)$$

Very similarly, in Datalog, the query can be expressed with the following program:

$$Q(z)\leftarrow F(x;y),F(y;z),\neg F(x;z).$$

▶ **Example 12.** Recall the query from Example 3 expressed in FLIF slightly differently as follows:

$$F(x;y_1)\,;F(x;y_2)\,;(F(y_1;z)\cap F(y_2;z))\,;(y_1\neq y_2)$$

This expression is already io-disjoint. The plan equivalent to this expression is:

$$\pi_z\sigma_{y_1\neq y_2}((In\bowtie F(x;y_1)\bowtie F(x;y_2)\bowtie F(y_1;z))\cap(In\bowtie F(x;y_1)\bowtie F(x;y_2)\bowtie F(y_2;z)))$$

where In is a relation name over $\{x\}$ providing input values.

In executable FO, the query will be expressed by the formula:

$$\exists y_1,y_2\, F(x;y_1)\wedge F(x;y_2)\wedge F(y_1;z)\wedge F(y_2;z)\wedge y_1\neq y_2$$

In Datalog, the program could be the following:

$$A(y_1,y_2,z)\leftarrow F(x;y_1),F(x;y_2),F(y_1;z),F(y_2;z).$$

$$B(y_1,y_2)\leftarrow A(y_1,y_2,z),y_1=y_2.$$

$$Q(z)\leftarrow A(y_1,y_2,z),\neg B(y_1,y_2).$$

4 Putting FLIF expressions in io-disjoint form

The main result of this paper is that arbitrary FLIF expressions can be put in io-disjoint form. We will first discuss the problem and its complications by means of illustrative examples. After that we formulate the precise theorem and give a constructive method to rewrite FLIF expressions into io-disjoint ones.

4.1 Examples

An obvious approach to obtaining an io-disjoint expression is to rename output variables. For example, we rewrite $R(x;x)$ to $R(x;y)$ and declare that the output value for x can now be found in slot y instead.

When applying this approach to the composition of two expressions, we must be careful, as an output of the first expression can be taken as input in the second expression. In that case, when renaming the output variable of the first expression, we must apply the renaming also to the second expression, but only on the input side. For example, $R(x;x)\,;S(x;x)$ is rewritten to $R(x;y)\,;S(y;z)$. Thus, the output x of the overall expression is renamed to z; the intermediate output x of the first expression is renamed to y, as is the input x of the second expression.

Obviously, we must also avoid variable clashes. For example, in $R(x;x)\,;S(y;y)$, when rewriting the subexpression $R(x;x)$, we should not use y to rename the output x to, as this variable is already in use in another subexpression.

Another subtlety arises in the rewriting of set operations. Consider, for example, the union $R(x;y) \cup S(x;z)$. As discussed in Example 8, this expression is not io-disjoint: the output variables are y and z, but these are also input variables, in addition to x. To make the expression io-disjoint, it does not suffice to simply rename y and z, say, to y_1 and z_1. We can, however, add assignments to both sides in such a way to obtain a formally io-disjoint expression:

$$R(x;y_1)\,;(z_1 := z) \;\cup\; S(x;z_1)\,;(y_1 := y).$$

The above trick must also be applied to intermediate variables. For example, consider $T(;) \cup (S(;y)\,;R(y;y))$. Note that T is a nullary relation. This expression is not io-disjoint with y being an input variable as well as an output variable. The second term is readily rewritten to $S(;y_1)\,;R(y_1;y_2)$ with y_2 the new output variable. Note that y_1 is an intermediate variable. The io-disjoint form becomes

$$T(;)\,;(y_1 := y)\,;(y_2 := y) \;\cup\; R(;y_1)\,;R(y_1;y_2).$$

In general, it is not obvious that one can always find a suitable variable to set intermediate variables from the other subexpression to. In our proof of the theorem we prove formally that this is always possible.

Expressions involving intersection can be rewritten with the aid of composition and equality test. For example the expression $R(x;y) \cap S(y;y)$ becomes

$$R(x;y_1)\,;S(y;y_2)\,;(y_1 = y_2).$$

The overall output y is renamed to y_1, but in the rewriting of the subexpression $S(y;y)$ we use an intermediate output variable y_2, then test that the two outputs are the same as required by the original expression. Note that this usage of intermediate variables is different from that used in the treatment of composition expressions. There, the intermediate variable was on the output side of the first subexpression and on the input side of the second subexpression; here, it is on the output side of the second subexpression.

An additional complication with intersection is that outputs that are not common to the lhs and rhs subexpressions of the intersection lose their status of output variable (Table 1), so must remain inertial for the overall expression. Hence, for the rewriting to have the desired semantics, we must add the appropriate equality tests. For example, the expression

$R(x, y; x, y) \cap S(x, z; x, z)$ has only x as an output, and x, y and z as inputs. Renaming the output x to x_1, it can be rewritten in io-disjoint form as

$$R(x, y; x_1, y_1) \,;\, (y_1 = y) \,;\, S(x, z; x_2, z_1) \,;\, (z_1 = z) \,;\, (x_2 = x_1).$$

A final complication occurs in the treatment of difference. Intermediate variables used in the rewriting must be reset to the same value in both subexpressions, since the difference operator is sensitive to the values of all variables. For example, let α be the expression $S(;x) \,;\, R(x; u, x) \;-\; T(;)$. We have $I(\alpha) = O(\alpha) = \{x, u\}$. Suppose we want to rename the outputs x and u to x_1 and u_1 respectively. As before, the subexpression on the lhs of the difference operator is rewritten to $S(;x_2) \,;\, R(x_2; u_1, x_1)$ introducing an intermediate variable x_2. Also as before, x_1 and u_1 need to be added to the rewriting of $T(;)$ which does not have x and u as outputs. But the new complication is that x_2 needs to be reset to a common value (we use x here) for the difference of the rewritten subexpressions to have the desired semantics. We thus obtain the overall rewriting

$$S(;x_2) \,;\, R(x_2; u_1, x_1) \,;\, (x_2 := x) \;-\; T(;) \,;\, (x_2 := x) \,;\, (u_1 := u) \,;\, (x_1 := x).$$

4.2 Statement of the theorem

As the overall idea behind the above examples was to rename the output variables, our aim is clearly the following theorem, with ρ playing the role of the renaming:[2]

▶ **Theorem 13.** *Let α be an FLIF expression and let ρ be a bijection from $O(\alpha)$ to a set of variables disjoint from* $\mathrm{var}(\alpha)$*. There exists an io-disjoint FLIF expression β such that*

1. $I(\beta) = I(\alpha)$*;*
2. $O(\beta) \supseteq \rho(O(\alpha))$*; and*
3. *for every interpretation D and every valuation ν_1, we have*

$$\{\nu_2|_{O(\alpha)} \mid (\nu_1, \nu_2) \in [\![\alpha]\!]_D\} = \{\nu_2 \circ \rho \mid (\nu_1, \nu_2) \in [\![\beta]\!]_D\}.$$

In the above theorem, we must allow $O(\beta)$ to be a superset of $\rho(O(\alpha))$ (rather than being equal to it), because we must allow the introduction of auxiliary (intermediate) variables. For example, let α be the expression $S(x;) - R(x; x)$. Note that $O(\alpha)$ is empty. Interpret S as holding bus stops and R as holding bus routes. Then α represents a module that takes as input x, and tests if x is a bus stop to where the bus would not return if we would take the bus at x. Assume, for the sake of contradiction, that there would exist an io-disjoint expression β as in the theorem, but with $O(\beta) = O(\alpha) = \emptyset$. Since $I(\beta)$ must equal $I(\alpha) = \{x\}$, the only variable occurring in β is x. In particular, β can only mention R in atomic subexpressions of the form $R(x; x)$, which is not io-disjoint. We are forced to conclude that β cannot mention R at all. Such an expression, however, can never be a correct rewriting of α. Indeed, let D be an interpretation for which $[\![\alpha]\!]_D$ is nonempty. Hence $[\![\beta]\!]_D$ is nonempty as well. Now let D' be the interpretation with $D'(S) = D(S)$ but $D'(R) = \emptyset$. Then $[\![\alpha]\!]_{D'}$ becomes clearly empty, but $[\![\beta]\!]_{D'} = [\![\beta]\!]_D$ remains nonempty since β does not mention R.

[2] We use $g \circ f$ for standard function composition ("g after f"). So, in the statement of the theorem, $\nu_2 \circ \rho : O(\alpha) \to \mathbb{V} : x \mapsto \nu_2(\rho(x))$.

4.3 Variable renaming

In the proof of our main theorem we need a rigorous way of renaming variables in FLIF expressions. The following lemma allows us to do this. It confirms that expressions behave under variable renamings as expected. The proof by structural induction is straightforward.

As to the notation used in the lemma, recall that $\mathbb{V}$ is the universe of variables. For a permutation θ of $\mathbb{V}$, and an expression α, we use $\theta(\alpha)$ for the expression obtained from α by replacing every occurrence of any variable x by $\theta(x)$.

▶ **Lemma 14** (Renaming Lemma). *Let α be an FLIF expression and let θ be a permutation of $\mathbb{V}$. Then for every interpretation D, we have*

$$(\nu_1, \nu_2) \in [\![\alpha]\!]_D \quad \Leftrightarrow \quad (\nu_1 \circ \theta, \nu_2 \circ \theta) \in [\![\theta(\alpha)]\!]_D.$$

4.4 Rewriting procedure

In order to be able to give a constructive proof of Theorem 13 by structural induction, a stronger induction hypothesis is needed. Specifically, to avoid clashes, we introduce a set W of forbidden variables. So we will actually prove the following statement:

▶ **Lemma 15.** *Let α be an FLIF expression, let W be a set of variables, and let ρ be a bijection from $O(\alpha)$ to a set of variables disjoint from $\mathrm{var}(\alpha)$. There exists an io-disjoint FLIF expression β such that*

1. *$I(\beta) = I(\alpha)$;*
2. *$O(\beta) \supseteq \rho(O(\alpha))$ and $O(\beta) - \rho(O(\alpha))$ is disjoint from W;*
3. *for every interpretation D and every valuation ν_1, we have*

$$\{\nu_2|_{O(\alpha)} \mid (\nu_1, \nu_2) \in [\![\alpha]\!]_D\} = \{\nu_2 \circ \rho \mid (\nu_1, \nu_2) \in [\![\beta]\!]_D\}.$$

We proceed to formally describe an inductive rewriting procedure to produce β from α as prescribed by the above lemma. The procedure formalizes and generalizes the situations encountered in the examples discussed in the previous section. The correctness of the method is proven in Section 5.

Terminology

A bijection from a set of variables X to another set of variables is henceforth called a *renaming of X*.

Module access

If α is of the form $M(\bar{x}; \bar{y})$, then β equals $M(\bar{x}; \rho(\bar{y}))$.

Variable assignment

If α is of the form $x := t$, then β equals $\rho(x) := t$.

Equality test

If α is an equality test, we can take β equal to α.

Nullary expressions

An expression α is called *nullary* if it contains no variables, i.e., $\mathrm{var}(\alpha)$ is empty. Trivially, for nullary α, the desired β can be taken to be α itself. We will consider this to be an extra base case for the induction.

Intersection

If α is of the form $\alpha_1 \cap \alpha_2$ then β equals $\beta_1 ; \gamma_1 ; \beta_2 ; \gamma_2 ; \eta$, where the constituent expressions are defined as follows.

- Let $W_1 = W \cup \mathrm{var}(\alpha)$ and let ρ_1 be a renaming of $O(\alpha_1)$ that is an extension of ρ such that the image of $\rho_1 - \rho$ is disjoint from W_1. By induction, there exists an io-disjoint rewriting of α_1 for W_1 and ρ_1; this yields β_1.
- Let $W_2 = W_1 \cup O(\beta_1)$ and let ρ_2 be a bijection from $O(\alpha_2)$ to a set of variables that is disjoint from W_2. By induction, there exists an io-disjoint rewriting of α_2 for W_2 and ρ_2; this yields β_2.
- γ_1 is the composition of all $(\rho_1(y) = y)$ for $y \in O(\alpha_1) - O(\alpha_2)$. If $O(\alpha_1) - O(\alpha_2)$ is empty, γ_1 can be dropped from the expression; this qualification applies to similar situations below.
- γ_2 is defined symmetrically.
- η is the composition of all $(\rho_1(y) = \rho_2(y))$ for $y \in O(\alpha)$.

Composition

If α is of the form $\alpha_1 ; \alpha_2$ then β equals $\beta_1 ; \theta(\beta_2)$, where the constituents are defined as follows.

- Let $W_2 = W \cup \mathrm{var}(\alpha) \cup \rho(O(\alpha_1))$, and let ρ_2 be the restriction of ρ to $O(\alpha_2)$. By induction, there exists an io-disjoint rewriting of α_2 for W_2 and ρ_2; this yields β_2.
- Let $W_1 = W \cup \mathrm{var}(\alpha)$, and let ρ_1 be a renaming of $O(\alpha_1)$ such that
  - on $O(\alpha_1) \cap O(\alpha_2) \cap I(\alpha_2)$, the image of ρ_1 is disjoint from $\mathrm{var}(\alpha) \cup O(\beta_2)$ as well as from the image of ρ;
  - elsewhere, ρ_1 agrees with ρ.

  By induction, there exists an io-disjoint renaming of α_1 for W_1 and ρ_1; this yields β_1.
- θ is the permutation of $\mathbb{V}$ defined as follows. For every $y \in I(\alpha_2) \cap O(\alpha_1)$, we have

  $$\theta(y) = \rho_1(y) \text{ and } \theta(\theta(y)) = y.$$

  Elsewhere, θ is the identity.

Union

If α is of the form $\alpha_1 \cup \alpha_2$ then β equals $(\beta_1 ; \gamma_1 ; \eta_1) \cup (\beta_2 ; \gamma_2 ; \eta_2)$ where the constituent expressions are defined as follows.

- Let $W_1 = W \cup \mathrm{var}(\alpha) \cup \rho(O(\alpha_2))$ and let ρ_1 be the restriction of ρ on $O(\alpha_1)$. By induction, there exists an io-disjoint rewriting of α_1 for W_1 and ρ_1; this yields β_1.
- Let $W_2 = W \cup \mathrm{var}(\alpha) \cup O(\beta_1)$ and let ρ_2 be the restriction of ρ on $O(\alpha_2)$. By induction, there exists an io-disjoint rewriting of α_2 for W_2 and ρ_2; this yields β_2.
- γ_1 is the composition of all $(\rho(y) := y)$ for $y \in O(\alpha_2) - O(\alpha_1)$, and γ_2 is defined symmetrically.

- If $O(\beta_2) - \rho_2(O(\alpha_2))$ (the set of "intermediate" variables in β_2) is empty, η_1 can be dropped from the expression. Otherwise, η_1 is the composition of all $(y := z)$ for $y \in O(\beta_2) - \rho(O(\alpha_2))$, with z a fixed variable chosen as follows.
  - **(a)** If $O(\beta_1)$ is nonempty, take z arbitrarily from there.
  - **(b)** Otherwise, take z arbitrarily from $\mathrm{var}(\alpha_2)$. We know $\mathrm{var}(\alpha_2)$ is nonempty, since otherwise α_2 would be nullary, so β_2 would equal α_2, and then $O(\beta_2)$ would be empty as well (extra base case), which is not the case.
- η_2 is defined symmetrically.

Difference

If α is of the form $\alpha_1 - \alpha_2$ then β equals $(\beta_1 \,;\, \gamma_1 \,;\, \eta_1 \,;\, \eta_2) - (\beta_2 \,;\, \gamma_2 \,;\, \eta_1 \,;\, \eta_2)$ where the constituent expressions are defined as follows.

- Let $W_1 = W \cup \mathrm{var}(\alpha)$ and let $\rho_1 = \rho$. By induction, there exists an io-disjoint rewriting of α_1 for W_1 and ρ; this yields β_1.
- Let $W_2 = W_1 \cup O(\beta_1)$ and let ρ_2 be a renaming of $O(\alpha_2)$ that agrees with ρ on $O(\alpha_1) \cap O(\alpha_2)$, such that the image of $\rho_2 - \rho_1$ is disjoint from W_2. By induction, there exists an io-disjoint rewriting of α_2 for W_2 and ρ_2; this yields β_2.
- γ_1 is the composition of all $(\rho_2(y) := y)$ for $y \in O(\alpha_2) - O(\alpha_1)$, and γ_2 is defined symmetrically.
- If $O(\beta_2) - \rho_2(O(\alpha_2))$ is empty, η_1 can be dropped from the expression. Otherwise, η_1 is the composition of all $(y := z)$ for $y \in O(\beta_2) - \rho_2(O(\alpha_2))$, with z a fixed variable chosen as follows.
  - **(a)** If $O(\alpha_1) \cap O(\alpha_2)$ is nonempty, take z arbitrarily from $\rho(O(\alpha_1) \cap O(\alpha_2))$.
  - **(b)** Otherwise, take z arbitrarily from $\mathrm{var}(\alpha_2)$ (which is nonempty by the same reasoning as given for the union case).
- η_2 is defined symmetrically.

4.5 Necessity of variable assignment

Our rewriting procedure intensively uses variable assignment. Is this really necessary? More precisely, suppose α itself does not use variable assignment. Can we still always find an io-disjoint rewriting β such that β does not use variable assignment either? Below, we answer this question negatively; in other words, the ability to do variable assignment is crucial for io-disjoint rewriting.

For our counterexample we work over the schema consisting of a nullary relation name S and a binary relation name T of input arity one. Let α be the expression $S(;) \cup T(x;x)$ and let ρ rename x to x_1. Note that our rewriting procedure would produce the rewriting

$$S(;) \,;\, (x_1 := x) \;\cup\; T(x;x_1),$$

indeed using a variable assignment $(x_1 := x)$ to ensure an io-disjoint expression.

For the sake of contradiction, assume there exists an expression β according to Theorem 13 that does not use variable assignment. Fix D to the interpretation where S is nonempty but T is empty. Then $[\![\alpha]\!]_D$ consists of all identical pairs of valuations. Take any valuation ν with $\nu(x) \neq \nu(x_1)$. Since $(\nu, \nu) \in [\![\alpha]\!]_D$, there should exist a valuation ν' with $\nu'(x_1) = \nu(x)$ such that $(\nu, \nu') \in [\![\beta]\!]_D$. Note that $\nu' \neq \nu$, since $\nu(x_1) \neq \nu(x)$. However, this contradicts the following two observations. Both observations are readily verified by induction. (Recall that D is fixed as defined above.)

1. For every expression β without variable assignments, either $[\![\beta]\!]_D$ is empty, or $[\![\beta]\!]_D = [\![\gamma]\!]_D$ for some expression γ that does not mention T and that has no variable assignments.
2. For every expression γ that does not mention T and that has no variable assignments, and any $(\nu_1, \nu_2) \in [\![\gamma]\!]_D$, we have $\nu_1 = \nu_2$.

5 Correctness proof

We prove that β constructed by the method described in Section 4.4 satisfies the statement of Lemma 15. The base cases are straightforwardly verified. For every inductive case, we need to verify several things:

Inputs: $I(\beta) = I(\alpha)$.

io-disjointness: Every subexpression of β, including β itself, must have disjoint inputs and outputs.

Outputs: $O(\beta) \supseteq \rho(O(\alpha))$.

No clashes: $O(\beta) - \rho(O(\alpha))$ is disjoint from W.

Completeness: For any interpretation D and $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$, we want to find ν such that $(\nu_1, \nu) \in [\![\beta]\!]_D$ and $\nu(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$.

Soundness: For any $(\nu_1, \nu_2) \in [\![\beta]\!]_D$, we want to find ν such that $(\nu_1, \nu) \in [\![\alpha]\!]_D$ and $\nu(y) = \nu_2(\rho(y))$ for $y \in O(\alpha)$.

Below we present here the proofs for composition and union, as these are the clearest. The proofs for intersection and difference are more intricate and will appear in the journal version of this paper.

5.1 Composition

Henceforth, for any expression δ, we will use the notation $\nu_1 \xrightarrow{\delta} \nu_2$ to indicate that $(\nu_1, \nu_2) \in [\![\delta]\!]_D$.

Inputs

We first analyze inputs and outputs for $\theta(\beta_2)$. Inputs pose no difficulty (note that $I(\beta_2) = I(\alpha_2)$). As to outputs, θ only changes variables in $I(\alpha_2)$ and β_2 is io-disjoint by induction, so θ has no effect on $O(\beta_2)$. Hence:

$$I(\theta(\beta_2)) = (I(\alpha_2) - O(\alpha_1)) \cup \rho_1(I(\alpha_2) \cap O(\alpha_1))$$
$$O(\theta(\beta_2)) = O(\beta_2)$$

Calculating $I(\beta)$, the part of $I(\theta(\beta_2))$ that is contained in $\rho_1(O(\alpha_1))$ disappears as $\rho_1(O(\alpha_1))$ is contained in $O(\beta_1)$. Also, $I(\beta_1) = I(\alpha_1)$ by induction. Thus $I(\beta) = I(\alpha_1) \cup (I(\alpha_2) - O(\alpha_1)) = I(\alpha)$ as desired.

Outputs

We verify:

$$\begin{aligned}\rho(O(\alpha)) &= \rho(O(\alpha_1)) \cup \rho(O(\alpha_2)) = \rho(O(\alpha_1) - O(\alpha_2)) \cup \rho(O(\alpha_2)) \\ &= \rho_1(O(\alpha_1) - O(\alpha_2)) \cup \rho_2(O(\alpha_2)) \subseteq O(\beta_1) \cup O(\beta_2) = O(\beta).\end{aligned}$$

io-disjointness

Expression β is io-disjoint since $O(\beta_1)$ and $O(\beta_2)$ are disjoint from $\mathrm{var}(\alpha)$ by construction. For subexpression $\theta(\beta_2)$, recall $I(\theta(\beta_2))$ and $O(\theta(\beta_2))$ as calculated above. The part contained in $I(\alpha_2)$ is disjoint from $O(\beta_2)$ since $I(\alpha_2) = I(\beta_2)$ and β_2 is io-disjoint by induction. We write the other part as $\rho_1(I(\alpha_2) \cap O(\alpha_1) \cap O(\alpha_2)) \cup \rho((I(\alpha_2) \cap O(\alpha_1)) - O(\alpha_2))$. The first term is disjoint from $O(\beta_2)$ by definition of ρ_1.

The second term is dealt with by the more general claim that $\rho(O(\alpha_1) - O(\alpha_2))$ *is disjoint from* $O(\beta_2)$. In proof, let $y \in O(\alpha_1) - O(\alpha_2)$ and assume for the sake of contradiction that $\rho(y) \in O(\beta_2)$. Then $\rho(y) \in O(\beta_2) - \rho(O(\alpha_2))$, which by induction is disjoint from W_2, which includes $\rho(O(\alpha_1))$. However, since $y \in O(\alpha_1)$, this is a contradiction.

No clashes

We have

$$\begin{aligned} O(\beta) - \rho(O(\alpha)) &= (O(\beta_1) \cup O(\beta_2)) - \rho(O(\alpha_1) \cup O(\alpha_2)) \\ &\subseteq (O(\beta_1) - \rho(O(\alpha_1)) \cup (O(\beta_2) - \rho(O(\alpha_2)). \end{aligned}$$

By induction, the latter two terms are disjoint from $W_1 \supseteq W$ and $W_2 \supseteq W$, respectively.

Completeness

Since $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$, there exists ν such that $\nu_1 \xrightarrow{\alpha_1} \nu \xrightarrow{\alpha_2} \nu_2$. By induction, there exists ν_3 such that $(\nu_1, \nu_3) \in [\![\beta_1]\!]_D$ and $\nu_3(\rho_1(y)) = \nu(y)$ for $y \in O(\alpha_1)$. Also by induction, there exists ν_4 such that $(\nu, \nu_4) \in [\![\beta_2]\!]_D$ and $\nu_4(\rho_2(y)) = \nu_2(y)$ for $y \in O(\alpha_2)$. By the Renaming Lemma (14), we have $(\nu \circ \theta, \nu_4 \circ \theta) \in [\![\theta(\beta_2)]\!]_D$.

We claim that ν_3 agrees with $\nu \circ \theta$ on $I(\theta(\beta))$. Recalling that the latter equals $(I(\alpha_2) - O(\alpha_1)) \cup \rho_1(I(\alpha_2) \cap O(\alpha_1))$, we verify this claim as follows.

- We begin by verifying that θ is the identity on $I(\alpha_2) - O(\alpha_1)$. Indeed, let $u \in I(\alpha_2) - O(\alpha_1)$. Note that θ is the identity outside $(I(\alpha_2) \cap O(\alpha_1)) \cup \rho_1(I(\alpha_2) \cap O(\alpha_1))$. Clearly u does not belong to the first term. Also u does not belong to the second term, since the image of ρ_1 is disjoint from $\mathrm{var}(\alpha)$.
- Now let $u \in I(\alpha_2) - O(\alpha_1)$. Then $\theta(u) = u$, so $(\nu \circ \theta)(u) = \nu(u)$. Now since

  $$\nu \xleftarrow{\alpha_1} \nu_1 \xrightarrow{\beta_1} \nu_3$$

  and u belongs neither to $O(\alpha_1)$ nor to $O(\beta_1)$ (as $O(\beta_1)$ is disjoint from $\mathrm{var}(\alpha)$), we get $\nu(u) = \nu_3(u)$.
- Let $u \in I(\alpha_2) \cap O(\alpha_1)$. Then $(\nu \circ \theta)(\rho_1(u)) = \nu(\theta(\theta(u))) = \nu(u)$. The latter equals $\nu_3(\rho_1(u))$ by definition of ν_3.

We can now apply input determinacy and obtain ν_5 such that $(\nu_3, \nu_5) \in [\![\theta(\beta_2)]\!]_D$ and ν_5 agrees with $\nu_4 \circ \theta$ on $O(\beta_2)$. It follows that $(\nu_1, \nu_5) \in [\![\beta]\!]_D$, so we are done if we can show that $\nu_5(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$. We distinguish two cases.

First, assume $y \in O(\alpha_2)$. Then $\nu_5(\rho(y)) = \nu_5(\rho_2(y)) = (\nu_4 \circ \theta)(\rho_2(y))$ by definition of ν_5. Now observe that $\theta(\rho_2(y)) = \rho_2(y)$. Indeed, $\rho_2(y)$ belongs to $O(\beta_2)$, while θ is the identity outside $(I(\alpha_2) \cap O(\alpha_1)) \cup \rho_1(I(\alpha_2) \cap O(\alpha_1))$. The first term is disjoint from $O(\beta_2)$ since $O(\beta_2)$ is disjoint from $\mathrm{var}(\alpha)$. The second term is disjoint from $O(\beta_2)$ as already shown in the io-disjointness proof. So, we obtain $\nu_4(\rho_2(y))$, which equals $\nu_2(y)$ by definition of ν_4.

Second, assume $y \in O(\alpha_1) - O(\alpha_2)$. Since $(\nu_3, \nu_5) \in [\![\theta(\beta_2)]\!]_D$ and $O(\theta(\beta_2)) = O(\beta_2)$ is disjoint from $\rho(O(\alpha_1) - O(\alpha_2))$ as seen in the disjointness proof, $\nu_5(\rho(y)) = \nu_3(\rho(y))$. Since $y \notin O(\alpha_2)$, we have $\nu_3(\rho(y)) = \nu_3(\rho_1(y))$, which equals $\nu(y)$ by definition of ν_3. Now $\nu(y) = \nu_2(y)$ since $(\nu, \nu_2) \in [\![\alpha_2]\!]_D$ and $y \notin O(\alpha_2)$.

Soundness

The proof for soundness is remarkably symmetrical to that for completeness. Such symmetry is not present in the proofs for the other operators. We cannot yet explain well why the symmetry is so present for composition.

Since $(\nu_1, \nu_2) \in [\![\beta]\!]_D$, there exists ν such that

$$\nu_1 \xrightarrow{\beta_1} \nu \xrightarrow{\theta(\beta_2)} \nu_2.$$

By induction, there exists ν_3 such that $(\nu_1, \nu_3) \in [\![\alpha_1]\!]_D$ and $\nu_3(y) = \nu(\rho_1(y))$ for $y \in O(\alpha_1)$. By the Renaming Lemma, we have $(\nu \circ \theta, \nu_2 \circ \theta) \in [\![\beta_2]\!]_D$ (note that $\theta^{-1} = \theta$). By induction, there exists ν_4 such that $(\nu \circ \theta, \nu_4) \in [\![\alpha_2]\!]_D$ and $\nu_4(y) = (\nu_2 \circ \theta)(\rho_2(y))$ for $y \in O(\alpha_2)$.

Using analogous reasoning as in the completeness proof, it can be verified that ν_3 agrees with $\nu \circ \theta$ on $I(\alpha_2)$. Hence, by input determinacy, there exists ν_5 such that $(\nu_3, \nu_5) \in [\![\alpha_2]\!]_D$ and ν_5 agrees with ν_4 on $O(\alpha_2)$. It follows that $(\nu_1, \nu_5) \in [\![\alpha]\!]_D$, so we are done if we can show that $\nu_5(y) = \nu_2(\rho(y))$ for $y \in O(\alpha)$. This is shown by analogous reasoning as in the completeness proof.

5.2 Union

Inputs

Let $\{i, j\} = \{1, 2\}$. We begin by noting:

$$I(\gamma_i) = O(\alpha_j) - O(\alpha_i)$$
$$O(\gamma_i) = \rho(O(\alpha_j) - O(\alpha_i))$$

Note that $I(\gamma_i)$, being a subset of $\mathrm{var}(\alpha)$, is disjoint from $O(\beta_i)$, so $I(\beta_i \,;\, \gamma_i)$ is simply $I(\beta_i) \cup I(\gamma_i)$. By induction, $I(\beta_i) = I(\alpha_i)$ and $O(\beta_i)$ contains $\rho(O(\alpha_i))$. Hence:

$$I(\beta_i \,;\, \gamma_i) = I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i))$$
$$O(\beta_i \,;\, \gamma_i) = O(\beta_i) \cup \rho(O(\alpha_j))$$

We next analyze η_i. Recall that this expression was defined by two cases.

(a) If $O(\beta_i)$ is nonempty, $I(\eta_i) \subseteq O(\beta_i)$.

(b) Otherwise, $I(\eta_i) \subseteq \mathrm{var}(\alpha_j)$. However, if $O(\beta_i)$ is empty then $O(\alpha_i)$ is too, so that $I(\alpha) = I(\alpha_i) \cup I(\alpha_j) \cup O(\alpha_j) = I(\alpha_i) \cup \mathrm{var}(\alpha_j)$. Hence, in this case, $I(\eta_i) \subseteq I(\alpha)$.

The output is the same in both cases:

$$O(\eta_i) = O(\beta_j) - \rho(O(\alpha_j))$$

Composing $\beta_i \,;\, \gamma_i$ with η_i, we continue with the two above cases.

(a) In this case $I(\eta_i)$ is contained in $O(\beta_i \,;\, \gamma_i)$, so $I(\beta_i \,;\, \gamma_i \,;\, \eta_i) = I(\beta_i \,;\, \gamma_i)$.

(b) In this case $I(\eta_i)$ is disjoint from $O(\beta_i \,;\, \gamma_i)$, and $I(\beta_i \,;\, \gamma_i \,;\, \eta_i)$ equals $I(\beta_i \,;\, \gamma_i)$ to which some element of $I(\alpha)$ is added.

In both cases, we can state that

$$I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i)) \subseteq I(\beta_i \,;\, \gamma_i \,;\, \eta_1) \subseteq I(\alpha).$$

For outputs, we have

$$O(\beta_i \,;\, \gamma_i \,;\, \eta_i) = O(\beta_1) \cup O(\beta_2).$$

The set of inputs of the final expression $\beta = (\beta_1 \,;\, \gamma_1 \,;\, \eta_1) \cup (\beta_2 \,;\, \gamma_2 \,;\, \eta_2)$ equals the union of inputs of the two top-level subexpressions, since these two subexpressions have the same outputs $(O(\beta_1) \cup O(\beta_2))$. Hence

$$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2)) \subseteq I(\beta) \subseteq I(\alpha).$$

Since the left expression equals $I(\alpha)$ by definition, we obtain that $I(\beta) = I(\alpha)$ as desired.

Outputs

From the above we have $O(\beta) = O(\beta_1) \cup O(\beta_2)$. Since $O(\beta_i) \supseteq \rho(O(\alpha_i))$ by induction, we obtain $O(\beta) \supseteq \rho(O(\alpha_1) \cup O(\alpha_2)) = \rho(O(\alpha))$ as desired.

io-disjointness

Let $i = 1, 2$. Expression γ_i is io-disjoint since the image of ρ is disjoint from $\mathrm{var}(\alpha)$. Then $\beta_i;\gamma_i$ is io-disjoint because both $O(\beta_i)$ and the image of ρ are disjoint from $\mathrm{var}(\alpha)$. For the same reason, $\beta_i \,;\, \gamma_i \,;\, \eta_i$ and β are io-disjoint. We still need to look at η_i. In case (b), $I(\eta_i) \subseteq I(\alpha)$ so io-disjointness follows again because $O(\beta_j)$ is disjoint from $\mathrm{var}(\alpha)$. In case (a), we look at $i = 1$ and $i = 2$ separately. For $i = 1$ we observe that $O(\eta_1) = O(\beta_2) - \rho(O(\alpha_2))$ is disjoint from W_2, which includes $O(\beta_1)$. For $i = 2$ we write $O(\beta_2) = \rho(O(\alpha_2)) \cup (O(\beta_2) - \rho(O(\alpha_2)))$. The first term is disjoint from $O(\eta_2) = O(\beta_1) - \rho(O(\alpha_1))$ since the latter is disjoint from W_1 which includes $\rho(O(\alpha_2))$. The second term is disjoint from $O(\beta_1)$ as we have just seen.

No clashes

We verify:

$$\begin{aligned} O(\beta) - \rho(O(\alpha)) &= (O(\beta_1) \cup O(\beta_2)) - \rho(O(\alpha_1) \cup O(\alpha_2)) \\ &\subseteq (O(\beta_1) - \rho(O(\alpha_1))) \cup (O(\beta_2) - \rho(O(\alpha_2))). \end{aligned}$$

By induction, both of the latter terms are disjoint from W, which confirms that there are no clashes.

Completeness

Assume $(\nu_1, \nu_2) \in [\![\alpha_1]\!]_D$; the reasoning for α_2 is analogous. By induction, there exists ν_3 such that $(\nu_1, \nu_3) \in [\![\beta_1]\!]_D$ and $\nu_3(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha_1)$.

Note that each of the expressions γ_i and η_i for $i = 1, 2$ is a composition of variable assignments. For any such expression δ and any valuation ν there always exists a unique ν' such that $(\nu, \nu') \in [\![\delta]\!]_D$ (even independently of D).

Now let

$$\nu_3 \xrightarrow{\gamma_1} \nu_4 \xrightarrow{\eta_1} \nu_5,$$

so that $(\nu_1, \nu_5) \in [\![\beta]\!]_D$. If we can show that $\nu_5(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$ we are done. Thereto, first note that η_1 does not change variables in $\rho(O(\alpha))$. Indeed, for $\rho(O(\alpha_2))$ this is obvious from $O(\eta_1) = O(\beta_2) - \rho(O(\alpha_2))$; for $\rho(O(\alpha_1))$ this follows because by induction, $O(\beta_2) - \rho(O(\alpha_2))$ is disjoint from W_2, which includes $O(\beta_1)$, which includes $\rho(O(\alpha_1))$. So, by $\nu_4 \xrightarrow{\eta_1} \nu_5$ we are down to showing that $\nu_4(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$. We distinguish two cases.

If $y \in O(\alpha_1)$, since $\nu_3 \xrightarrow{\gamma_1} \nu_4$ and γ_1 does not change variables in $\rho(O(\alpha_1))$, we have $\nu_4(\rho(y)) = \nu_3(\rho(y))$, which equals $\nu_2(y)$ by definition of ν_3.

If $y \in O(\alpha_2) - O(\alpha_1)$, then $\nu_4(\rho(y)) = \nu_3(y)$ by $\nu_3 \xrightarrow{\gamma_1} \nu_4$. Now since

$$\nu_3 \xleftarrow{\beta_1} \nu_1 \xrightarrow{\alpha_1} \nu_2$$

and $y \notin O(\beta_1) \cup O(\alpha_1)$, we get $\nu_3(y) = \nu_2(y)$ as desired. (The reason for $y \notin O(\beta_1)$ is that by induction, $O(\beta_1)$ is disjoint from W_1 which includes $\mathrm{var}(\alpha)$.)

Soundness

Assume $(\nu_1, \nu_2) \in [\![\beta_1 \,;\, \gamma_1 \,;\, \eta_1]\!]_D$; the reasoning for $\beta_2 \,;\, \gamma_2 \,;\, \eta_2$ is analogous. Then there exist ν_3 and ν_4 such that

$$\nu_1 \xrightarrow{\beta_1} \nu_3 \xrightarrow{\gamma_1} \nu_4 \xrightarrow{\eta_1} \nu_2. \qquad (*)$$

By induction, there exists ν such that $(\nu_1, \nu) \in [\![\alpha_1]\!]_D \subseteq [\![\alpha]\!]_D$ and $\nu(y) = \nu_3(\rho(y))$ for $y \in O(\alpha_1)$. As observed in the completeness proof, γ_1 and η_1 do not touch variables in $\rho(O(\alpha_1))$. Hence by $(*)$ also $\nu(y) = \nu_2(\rho(y))$ for $y \in O(\alpha_1)$.

If we can show the same for $y \in O(\alpha_2) - O(\alpha_1)$, we have covered all $y \in O(\alpha)$ and we are done. This is verified as follows. By inertia, we have $\nu(y) = \nu_1(y) = \nu_3(y)$, the latter equality because $O(\beta_1)$ is disjoint from $\mathrm{var}(\alpha)$. From $\nu_3 \xrightarrow{\gamma_1} \nu_4$ we have $\nu_3(y) = \nu_4(\rho(y))$. Now the latter equals $\nu_2(\rho(y))$ since $\nu_4 \xrightarrow{\eta_1} \nu_2$ and η_1 does not touch variables in $\rho(O(\alpha_2))$.

6 Conclusion

We have shown how to rewrite arbitrary FLIF expressions into io-disjoint ones. Our rewriting procedure is polynomial, as is readily verified from the description given in Section 4.4.

While the problem of "making programs io-disjoint" was, in this paper, interpreted in a specific manner according to the definitions of the FLIF language, there seems to be a more general quality about the problem that we have not been able to articulate. Of course, many examples of techniques that seem superficially similar can be cited, such as renaming bound variables in logic, or alpha-conversion in lambda calculus. But input and output as interpreted in this paper have a definite dynamic (one could say procedural) meaning which is lacking in such examples. So, while we are not aware about known results in logic or the foundations of programming languages that are technically related to our main result or our proof techniques, we would love to find out about them if they exist.

There are many interesting topics for further research on FLIF, and on LIF (Logic of Information Flows [17, 18, 1, 2]) in general. Some are already discussed in the cited papers; in closing this paper we list some more.

1. In our rewriting technique, there are really three, not two, categories of variables into play: inputs, outputs, and intermediate variables. It would be interesting to develop the framework further so that intermediate variables get a full treatment alongside inputs and outputs.

2. Investigate io-disjointness when FLIF is extended with additional operators, notably converse and transitive closure. We remark, however, that transitive closure seems to be at odds with io-disjointness. Indeed, if α is io-disjoint then the transitive closure of $[\![\alpha]\!]_D$ equals $[\![\alpha]\!]_D$ itself.
3. Delineate useful instances of LIF for which the problems of checking whether a variable is a semantic input, or output, are decidable.
4. Apply FLIF in practice. In this regard, the language could be made more practical by extending it with an explicit construct for hiding variables, as found, e.g., in process calculi [13], graph expressions [6], and graph logic [8].

References

1 H. Aamer, B. Bogaerts, D. Surinx, E. Ternovska, and J. Van den Bussche. Executable first-order queries in the logic of information flows. In C. Lutz and J.C. Jung, editors, *Proceedings 23rd International Conference on Database Theory*, volume 155 of *Leibniz International Proceedings in Informatics*, pages 4:1–4:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

2 H. Aamer, B. Bogaerts, D. Surinx, E. Ternovska, and J. Van den Bussche. Inputs, outputs, and composition in the logic of information flows. In D. Calvanese, E. Erdem, and M. Thielscher, editors, *Proceedings 17th International Conference on Principles of Knowledge Representation and Reasoning*. IJCAI Organization, 2020.

3 H. Andréka, S. Givant, and I. Németi. *Decision problems for equational theories of relation algebras*, volume 126 of *Memoirs of the American Mathematical Society*. American Mathematical Society, 1997.

4 R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, 50(5):68:1–68:40, 2017.

5 R. Angles, P. Barceló, and G. Rios. A practical query language for graph DBs. In L. Bravo and M. Lenzerini, editors, *Proceedings 7th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 1087 of *CEUR Workshop Proceedings*, 2013.

6 M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20:83–127, 1987.

7 M. Benedikt, J. Leblay, B. ten Cate, and E. Tsamoura. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Morgan & Claypool, 2016.

8 L. Cardelli, Ph. Gardner, and G. Ghelli. A spatial logic for querying graphs. In P. Widmayer et al., editors, *Proceedings 29th International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer, 2002.

9 G.H.L. Fletcher, M. Gyssens, D. Leinders, D. Surinx, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. *Information Sciences*, 298:390–406, 2015.

10 L. Libkin, W. Martens, and D. Vrgoč. Quering graph databases with XPath. In W.-C. Tan et al., editors, *Proceedings 16th International Conference on Database Theory*, pages 129–140. ACM, 2013.

11 V. Lifschitz. Formal theories of action (preliminary report). In J.P. McDermott, editor, *Proceedings 10th International Joint Conference on Artificial Intelligence*, pages 966–972. Morgan Kaufmann, 1987.

12 J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

13 R. Milner. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, 1999.

14 A. Nash and B. Ludäscher. Processing first-order queries under limited access patterns. In *Proceedings 23th ACM Symposium on Principles of Database Systems*, pages 307–318, 2004.

15 J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.

16 D. Surinx, G.H.L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs using transitive closure. *Logic Journal of the IGPL*, 23(5):759–788, 2015.

17 E. Ternovska. Recent progress on the algebra of modular systems. In J.L. Reutter and D. Srivastava, editors, *Proceedings 11th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 1912 of *CEUR Workshop Proceedings*, 2017.

18 E. Ternovska. An algebra of modular systems: static and dynamic perspectives. In A. Herzig and A. Popescu, editors, *Frontiers of Combining Systems: Proceedings 12th FroCos*, volume 11715 of *Lecture Notes in Artificial Intelligence*, pages 94–111. Springer, 2019.

Conjunctive Queries: Unique Characterizations and Exact Learnability

Balder ten Cate
Google, Mountain View, CA, USA

Victor Dalmau
Universitat Pompeu Fabra, Barcelona, Spain

Abstract

We answer the question of which conjunctive queries are uniquely characterized by polynomially many positive and negative examples, and how to construct such examples efficiently. As a consequence, we obtain a new efficient exact learning algorithm for a class of conjunctive queries. At the core of our contributions lie two new polynomial-time algorithms for constructing frontiers in the homomorphism lattice of finite structures. We also discuss implications for the unique characterizability and learnability of schema mappings and of description logic concepts.

2012 ACM Subject Classification Theory of computation → Machine learning theory; Theory of computation → Logic; Information systems → Query languages

Keywords and phrases Conjunctive Queries, Homomorphisms, Frontiers, Unique Characterizations, Exact Learnability, Schema Mappings, Description Logic

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.9

Related Version *Full Version*: `https://arxiv.org/pdf/2008.06824.pdf`

Funding *Victor Dalmau*: Victor Dalmau was supported by MICCIN grants TIN2016-76573-C2-1P and PID2019-109137GB-C22.

Acknowledgements This paper largely grew out of discussions at Dagstuhl Seminar 19361 ("Logic and Learning") in Sept. 2019. We thank Carsten Lutz and Phokion Kolaitis for helpful discussions.

1 Introduction

Conjunctive queries (CQs) are an extensively studied database query language and fragment of first-order logic. They correspond precisely to Datalog programs with a single non-recursive rule. In this paper, we study two problems related to CQs. The first problem is concerned with the existence and constructability of unique characterizations. *For which CQs q is it the case that q can be characterized (up to logical equivalence) by its behavior on a small set of data examples? And, when such a set of data examples exists, can it be constructed efficiently?* The second problem pertains to *exact learnability* of CQs in an interactive setting where the learner has access to a "membership oracle" that, given any database instance and a tuple of values, answers whether the tuple belongs to the answer of the goal CQ (that is, the hidden CQ that the learner is trying to learn). We can think of the membership oracle as a black-box, compiled version of the goal query, which the learner can execute on any number of examples. The task of the learner, then, is to reverse engineer the query based on the observed behavior.

Note that these two problems (unique characterizability and exact learnability) are closely related to each other: a learner can identify the goal query with certainty, only when the set of examples that it has seen so far constitutes a unique characterization of the goal query.

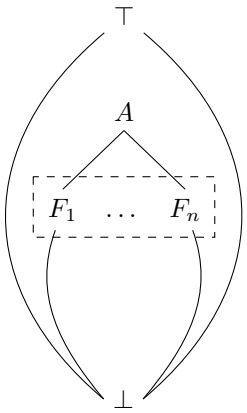


Figure 1 A frontier in the homomorphism lattice of structures.

▶ **Motivating Example 1.** *This example, although stylized and described at a high level, aims to convey an important use case that motivated the present work. The* Google Knowledge Graph *is a large database of entities and facts, gathered from a variety of sources. It is used to enhance the search engine's results for queries such as "where was Barack Obama born" with factual information in the form of* knowledge panels *[7] When a query triggers a specific knowledge panel, this may be the result of various different triggering and fulfillment mechanisms, which may involve a combination of structured queries to the knowledge graph, hard-coded business logic (in a Turing-complete language), and machine learned models. This makes it difficult to understand interactions between knowledge panels (e.g., whether the two knowledge panels are structurally equivalent or one subsumed by the other). If a declarative specification of (an approximation of) the triggering and fulfillment logic for a knowledge panel can be constructed programmatically, specified in a sufficiently restrictive formalism such as Datalog rules, this provides an avenue to the above, and other relevant static analysis tasks. The* efficient exact learnability with membership queries *that we study in this paper, can be viewed as an idealized form of such a programmatic approach, where the membership oracle is the existing, black box, implementation of the knowledge panel, and the learner must produce a CQ that exactly captures it.*

As it turns out, the above problems about CQs are intimately linked to fundamental properties of the homomorphism lattice of finite structures. In particular, the existence of a unique characterization for a CQ can be reduced to the existence of a *frontier* in the homomorphism lattice for an associated structure A, where, by a "frontier" for A, we mean a finite set of structures $F_1, \ldots, F_n$ that cover precisely the set of structures homomorphically strictly weaker than A, that is, such that $\{B \mid B \overset{\rightarrow}{\not\leftarrow} A\} = \bigcup_i \{B \mid B \to F_i\}$ (cf. Figure 1).

Known results [16, 1] imply that not every finite structure has such a frontier, and, moreover, a finite structure has a frontier if and only if the structure (modulo homomorphic equivalence) satisfies a structural property called *c-acyclicity*. These known results, however, are based on exponential constructions, and no polynomial algorithms for constructing frontiers were previously known.

▶ **Main Contribution 1** (Polynomial-time algorithms for constructing frontiers)**.** *We show that, for c-acyclic structures, a frontier can in fact be computed* in polynomial time*. More specifically, we present two polynomial-time algorithms. The first algorithm takes any c-acyclic structure and produces a frontier consisting of structures that are themselves not necessarily*

c-acyclic. Indeed, we show that this is unavoidable. The second algorithm applies to a more restricted class of acyclic structures and yields a frontier consisting entirely of structures belonging to the same class (that is, the class of structures in question is frontier-closed).

We use these to obtain new results on the existence and efficient constructability of unique characterizations for CQs:

▶ **Main Contribution 2** (Polynomial Unique Characterizations for Conjunctive Queries)**.** *We show that a CQ is uniquely characterizable by polynomially many examples, precisely if (modulo logical equivalence) it is c-acyclic. Furthermore, for c-acyclic CQs, a uniquely characterizing set of examples can be constructed in polynomial time. In the special case of unary, acyclic, connected CQs, a uniquely characterizing set of examples can be constructed consisting entirely of queries from the same class.*

Using the above results as a stepping stone, we obtain a polynomial-time exact learning algorithm for the class of c-acyclic CQs.

▶ **Main Contribution 3** (Polynomial-Time Learnability with Membership Queries)**.** *We show that c-acyclic CQs are efficiently exactly learnable in Angluin's model of exact learnability with membership queries [2].*

The above results are optimal, because, as we mentioned above, exact learnability with membership queries, requires the existence of a finite uniquely characterizing set of examples, which, in turn, requires the existence of a frontier, and frontiers exist (up to homomorphic equivalence) only for c-acyclic structures.

Although our primary interest is in conjunctive queries, we show that our results also have implications for *schema mappings* and *description logic concepts*:

▶ **Main Contribution 4** (Schema Mappings and Description Logic Concepts)**.** *As a further corollary to the above, we obtain a number of results regarding the existence of polynomial unique characterizations, as well as exact learnability, for LAV ("Local-As-View") schema mappings and for description logic concepts for the lightweight description logic $\mathcal{ELI}$.*

Outline

Section 2 reviews basic facts and definitions. In Section 3, we present our two new polynomial-time algorithms for constructing frontiers for finite structures with designated elements. We also review a result by [27], which implies the existence of (not necessarily polynomially computable) frontiers w.r.t. classes of structures of bounded expansion. In Section 4, we apply these algorithms to show that a CQ is uniquely characterizable by polynomially many examples, precisely if (modulo logical equivalence) it is c-acyclic. Furthermore, for c-acyclic CQs, a uniquely characterizing set of examples can be constructed in polynomial time. In the special case of unary, acyclic, connected CQs, a uniquely characterizing set of examples can be constructed consisting entirely of queries from the same class. In Section 5, we further build on these results, and we study the exact learnability of CQs. Section 6 presents applications to schema mappings and description logic concepts.

2 Preliminaries

Schemas, Structures, Homomorphisms, Cores. A *schema* (or, relational signature) is a finite set of relation symbols $\mathcal{S} = \{R_1, \ldots, R_n\}$, where each relation R_i as an associated arity $\text{arity}(R_i) \geq 1$. For $k \geq 0$, by a *structure over $\mathcal{S}$ with k distinguished elements* we

will mean a tuple $(A, a_1, \ldots, a_k)$, where $A = (\mathrm{dom}(A), R_1^A, \ldots, R_n^A)$ is a finite structure (in the traditional, model-theoretic sense) over the schema $\mathcal{S}$, and $a_1, \ldots, a_k$ are elements of the domain of A. Note that all structures, in this paper, are assumed to be finite, and we will drop the adjective "finite". By a *fact* of a structure A we mean an expression of the form $R(a_1, \ldots, a_n)$ where the tuple $(a_1, \ldots, a_n)$ belongs to the relation R in A. Given two structures $(A, \mathbf{a})$ and $(B, \mathbf{b})$ over the same schema, where $\mathbf{a} = a_1, \ldots a_k$ and $\mathbf{b} = b_1, \ldots, b_k$, a *homomorphism* $h : (A, \mathbf{a}) \to (B, \mathbf{b})$ is a map h from the domain of A to the domain of B, such that h preserves all facts, and such that $h(a_i) = b_i$ for $i = 1 \ldots k$. When such a homomorphism exists, we will also say that $(A, \mathbf{a})$ "homomorphically maps to" $(B, \mathbf{b})$ and we will write $(A, \mathbf{a}) \to (B, \mathbf{b})$. We say that $(A, \mathbf{a})$ and $(B, \mathbf{b})$ are *homomorphically equivalent* if $(A, \mathbf{a}) \to (B, \mathbf{b})$ and $(B, \mathbf{b}) \to (A, \mathbf{a})$.

A structure is said to be a *core* if there is no homomorphism from the structure in question to a proper substructure [20]. It is known [20] that every structure $(A, \mathbf{a})$ has a substructure to which it is homomorphically equivalent and that is a core. This substructure, moreover, is unique up to isomorphism, and it is known as *the core of* $(A, \boldsymbol{a})$.

Fact Graph, FG-Connectedness, FG-Disjoint Union. The *fact graph* of a structure $(A, \mathbf{a})$ is the undirected graph whose nodes are the facts of A, and such that there is an edge between two distinct facts if they share a non-designated element, i.e., there exists an element b of the domain of A that is distinct from the distinguished elements $\mathbf{a}$, such that b occurs in both facts. We say that $(A, \mathbf{a})$ is *fg-connected* if the fact graph is connected. A *fg-connected component* of $(A, \mathbf{a})$ is a maximal fg-connected substructure $(A', \mathbf{a})$ of $(A, \mathbf{a})$. If $(A_1, \mathbf{a})$ and $(A_2, \mathbf{a})$ are structures with the same distinguished elements, and whose domains are otherwise (except for these distinguished elements) disjoint, then the union $(A_1 \cup A_2, \mathbf{a})$ of these two structures will be called a *fg-disjoint union* and will be denoted as $(A_1, \mathbf{a}) \uplus (A_2, \mathbf{a})$. The same construction naturally extends to finite sets of structures. It is easy to see that every structure $(A, \mathbf{a})$ is equal to the fg-disjoint union of its fg-connected components. See also [15, 34], where fg-connected components are called *fact blocks*.

Direct Product, Homomorphism Lattice. Given two structures $(A, \mathbf{a})$ and $(B, \mathbf{b})$ over the same schema, where $\mathbf{a} = a_1, \ldots a_k$ and $\mathbf{b} = b_1, \ldots, b_k$, the *direct product* $(A, \mathbf{a}) \times (B, \mathbf{b})$ is defined, as usual, as $((A \times B), \langle a_1, b_1 \rangle, \ldots, \langle a_k, b_k \rangle)$, where the domain of $A \times B$ is the Cartesian product of the domains of A and B, and where the facts of $A \times B$ are all facts $R(\langle c_1, d_1 \rangle, \ldots, \langle c_n, d_n \rangle)$ for which it holds that $R(c_1, \ldots, c_n)$ is a fact of A and $R(d_1, \ldots, d_n)$ is a fact of B. The direct product of a finite collection of structures is defined analogously.

For a fixed schema $\mathcal{S}$ and $k \geq 0$, the collection of (homomorphic-equivalence classes of) structures over $\mathcal{S}$ with k distinguished elements, ordered by homomorphism, forms a lattice, where the above *direct product* operation is the meet operation. In particular, $(A, \mathbf{a}) \times (B, \mathbf{b})$ homomorphically maps to both $(A, \mathbf{a})$ and $(B, \mathbf{b})$, and a structure $(C, \mathbf{c})$ homomorphically maps to $(A, \mathbf{a}) \times (B, \mathbf{b})$ if and only if it homomorphically maps to both $(A, \mathbf{a})$ and $(B, \mathbf{b})$. The join operation of the lattice is a little more tedious to define, and we only sketch it here, as it is not used in the remainder of the paper. When two structures have the same isomorphism type of distinguished elements, their join is simply the fg-disjoint union as defined earlier. In the general case, one must first compute the smallest equivalence relation over the index set of the distinguished elements that refines the equivalence relations induced by both structures, and factor both structures through, before taking their fg-disjoint union.

For structures without designated elements, this lattice has been studied extensively (cf. for instance [21, 29]). The above exposition shows how to lift some of the basic definitions and constructions, in the appropriate way, to structures with distinguished elements.

Incidence Graph, C-Acyclicity. Given a structure $(A, \mathbf{a})$, the *incidence graph* of A is the bipartite multi-graph containing all elements of the domain of A as well as all facts of A, and an edge (a, f) whenever a is an element and f is a fact in which a occurs. Note that if an element occurs more than once in the same fact, the incidence graph contains multiple edges. $(A, \mathbf{a})$ is said to be *c-acyclic* if every cycle in its incidence graph contains at least one distinguished elements, i.e., at least one elements in $\mathbf{a}$. In particular, this means that no non-designated element occurs twice in the same fact. The concept of c-acyclicity was first introduced in [1] in the study of unique characterizability of GAV schema mappings (cf. Section 6 for more details). A straightforward dynamic-programming argument shows:

▶ **Proposition 2.1.** *For c-acyclic structures $(A, \boldsymbol{a})$ and $(B, \boldsymbol{b})$ (over the same schema and with the same number of distinguished elements), we can test in polynomial time whether $(A, \boldsymbol{a}) \to (B, \boldsymbol{b})$. The core of a c-acyclic structure can be computed in polynomial time.*

In the case without designated elements, c-acyclicity simply means that the incidence graph is acyclic, a condition better known as *Berge acyclicity* in the database theory literature [14].

C-Connectedness. We say that a structure $(A, \mathbf{a})$ is *c-connected* if every connected component of its incidence graph contains at least one designated element. Note that this condition is only meaningful for $k > 0$, and that it differs subtly from the condition of fg-connectedness we defined above. For example, the structure consisting of the facts $R(a_1, a_2)$ and $S(a_2, a_1)$ with distinguished elements a_1, a_2, is c-connected but is *not* fg-connected. For any structure $(A, \mathbf{a})$, we denote by $(A, \mathbf{a})^{\text{reach}}$ the (unique) maximal c-connected substructure, that is, the substructure containing everything reachable from the distinguished elements.

▶ **Proposition 2.2.** *If $(A, \boldsymbol{a})$ is c-connected, then $(A, \boldsymbol{a}) \to (B, \boldsymbol{b})^{reach}$ iff $(A, \boldsymbol{a}) \to (B, \boldsymbol{b})$.*

Conjunctive Queries. Let $k \geq 0$. A k-ary *conjunctive query (CQ)* q over a schema $\mathcal{S}$ is an expression of the form $q(\mathbf{x}) \text{ :- } \alpha_1 \wedge \cdots \wedge \alpha_n$ where $\mathbf{x} = x_1, \ldots, x_k$ is a sequence of variables, and where each α_i is an atomic formula using a relation from $\mathcal{S}$. Note that α_i may use variables from $\mathbf{x}$ as well as other variables. In addition, it is required that each variable in $\mathbf{x}$ occurs in at least one conjunct α_i. This requirement is referred to as the *safety* condition.

Note that, for simplicity, this definition of CQ does not allow the use of constants. Many of the results in this paper, however, can be extended in a straightforward way to CQs with a fixed finite number of constants (which can be simulated using additional free variables).

If A is a structure over the same schema as q, we denote by $q(A)$ the set of all k-tuples of values that satisfy the query q in A. We write $q \subseteq q'$ if q and q' are queries over the same schema, and of the same arity, and $q(A) \subseteq q'(A)$ holds for all structures A. We say that q and q' are *logically equivalent* if $q \subseteq q'$ and $q' \subseteq q$ both hold. We refer to any textbook on database theory for a more detailed exposition of the semantics of CQs.

There is a well-known correspondence between k-ary CQs over a schema $\mathcal{S}$ and structures over $\mathcal{S}$ with k distinguished elements. In one direction, we can associate to each k-ary CQ $q(\mathbf{x})$ over the schema $\mathcal{S}$ a corresponding structure over $\mathcal{S}$ with k distinguished elements, namely $\widehat{q} = (A_q, \mathbf{x})$, where the domain of A_q is the set of variables occurring in q, and the facts of A_q are the conjuncts of q. We will call this structure $\widehat{q}$ the *canonical structure* of q. Note that every distinguished element of $\widehat{q}$ occurs in at least one fact, as follows from the safety condition of CQs. Conversely, consider any structure $(A, \mathbf{a})$, with $\mathbf{a} = a_1, \ldots, a_k$, such that every distinguished element a_i occurs in at least one fact of A. We can associate to $(A, \mathbf{a})$ a k-ary *canonical CQ*, namely the CQ that has a variable x_a for every value a in the domain of A occurring in at least one fact, and a conjunct for every fact of A.

By the classic *Chandra-Merlin Theorem* [10], a tuple $\mathbf{a}$ belongs to $q(A)$ if and only if there is a homomorphism from $\widehat{q}$ to $(A, \mathbf{a})$; and $q \subseteq q'$ holds if and only if there is a homomorphism from $\widehat{q'}$ to $\widehat{q}$. Finally, q and q' are logically equivalent if and only if $\widehat{q}$ and $\widehat{q'}$ are homomorphically equivalent.

Exact Learning Models, Conjunctive Queries as a Concept Class. Informally, an *exact learning algorithm* is an algorithm that identifies an unknown goal concept by asking a number of queries about it. The queries are answered by an oracle that has access to the goal concept. This model of learning was introduced by Dana Angluin, cf. [2]. In this paper, we consider the two most extensively studied kinds of oracle queries: *membership queries* and *equivalence queries*. We will first review basic notions from computational learning theory, such as the notion of a *concept*, and then explain what it means for a concept class to be *efficiently exactly learnable with membership and/or equivalence queries*.

Let X be a (possibly infinite) set of *examples*. A *concept over* X is a function $c : X \to \{0, 1\}$, and a *concept class* $\mathcal{C}$ is a collection of such concepts. We say that $x \in X$ is a *positive example* for a concept c if $c(x) = 1$, and that x is a *negative example* for c if $c(x) = 0$.

Conjunctive queries (over a fixed schema $\mathcal{S}$ and with a fixed arity k) are a particular example of such a concept class, where the example space is the class of all structures over $\mathcal{S}$ with k distinct elements, and where an example $(A, \mathbf{a})$ is labeled as positive if the tuple $\mathbf{a}$ belongs to $q(A)$, and negative otherwise.

It is always assumed that concepts are specified using some representation system so that one can speak of the length of the specification of a concept. More formally, a *representation system for* $\mathcal{C}$ is a string language $\mathcal{L}$ over some finite alphabet, together with a surjective function $r : \mathcal{L} \to \mathcal{C}$. By the *size* of a concept $c \in \mathcal{C}$, we will mean the length of the smallest representation. Similarly, we assume a representation system, with a corresponding notion of length, for the examples in X. When there is no risk of confusion, we may conflate concepts (and examples) with their representations.

Specifically, for us, when it comes to *structures*, any natural choice of representation will do; we only assume that the length of the specification of a structure (for a fixed schema) is polynomial in the domain size, the number of facts and the number of distinguished elements. Likewise for *CQs*, we assume that the length of the representation of a CQ is polynomial in that of its canonical structure.

For every concept c, we denote by MEM_c the *membership oracle* for c, that is, the oracle that takes as input an example x and returns its label, $c(x)$, according to c. Similarly, for every concept $c \in \mathcal{C}$, we denote by EQ_c, the *equivalence oracle* for c, that is, the oracle that takes as input the representation of a concept h and returns "yes", if $h = c$, or returns a counterexample x otherwise (that is, an example x such that $h(x) \neq c(x)$). An *exact learning algorithm with membership and/or equivalence queries* for a concept class C is an algorithm `alg` that takes no input but has access to the membership oracle and/or equivalence oracle for an unknown *goal concept* $c \in C$.[1] The algorithm `alg` must terminate after finite amount of time and output (some representation of) the goal concept c. This notion was introduced by Angluin [2], who also introduced the notion of a *polynomial-time* exact learning algorithm. We say that an exact learning algorithm `alg` with membership and/or equivalence queries *runs in polynomial time* if there exists a two-variable polynomial $p(n, m)$ such that at any

[1] It is common in the learning theory literature to assume that the learning algorithm is given an upper bound on the size of the goal concept as input. However, it turns out that such an assumption is not needed for any of our positive results concerning learnability.

point during the run of the algorithm, the time used by `alg` up to that point (counting one step per oracle call) is bounded by $p(n, m)$, where n is the size of the goal concept and m the size of the largest counterexample returned by calls to the equivalence oracle up to that point in the run ($m = 0$ if no equivalence queries have been used). A concept class $\mathcal{C}$ is *efficiently exactly learnable with membership and/or equivalence queries* if there is an exact learning algorithm with membership and/or equivalence queries for $\mathcal{C}$ that runs in polynomial time.

There is a delicate issue about this notion of polynomial time that we now discuss. One might be tempted to relax the previous definition by requiring merely that the total running time is bounded by $p(n, m)$. However, this change in the definition would give rise to a *wrong* notion of a polynomial-time algorithms in this context by way of a loophole in the definition. Indeed, under this change, one could design a learning algorithm that, in a first stage, identifies the goal hypothesis by (expensive) exhaustive search and that, once this is achieved, forces – by asking equivalence queries with appropriate modification of the goal concept – the equivalence oracle to return large counterexamples that would make up for the time spent during the exhaustive search phase.

3 Frontiers in the homomorphism lattice of structures

In this section, we define frontiers and discuss their relationships to gaps and (restricted) homomorphism dualities. We present two polynomial-time methods for constructing frontiers. For the applications in the next sections, it is important to consider structures with designated elements. These designated elements, intuitively, correspond to the free variables of a CQ.

▶ **Definition 3.1.** *Fix a schema and $k \geq 0$, and let $\mathcal{C}$ be a class of structures with k designated elements and let $(A, \boldsymbol{a})$ be a structure with k designated elements as well. A* frontier *for $(A, \boldsymbol{a})$ w.r.t. $\mathcal{C}$, is a set of structures F such that*

1. *$(B, \boldsymbol{b}) \to (A, \boldsymbol{a})$ for all $(B, \boldsymbol{b}) \in F$.*
2. *$(A, \boldsymbol{a}) \not\to (B, \boldsymbol{b})$ for all $(B, \boldsymbol{b}) \in F$.*
3. *For all $(C, \boldsymbol{c}) \in \mathcal{C}$ with $(C, \boldsymbol{c}) \to (A, \boldsymbol{a})$ and $(A, \boldsymbol{a}) \not\to (C, \boldsymbol{c})$, we have that $(C, \boldsymbol{c}) \to (B, \boldsymbol{b})$ for some $(B, \boldsymbol{b}) \in F$.*

See Figure 1 for a graphical depiction of a frontier.

The notion of a frontier is closely related to that of a gap pair. A pair of structures (B, A) with $B \to A$ is said to be a *gap pair* if $A \not\to B$, and every structure C satisfying $B \to C$ and $C \to A$ is homomorphically equivalent to either B or A [30]. The same concept applies to structures with designated elements. It is easy to see that any frontier for a structure A must contain (modulo homomorphic equivalence) all structures B such that (B, A) is a gap pair.

▶ **Example 3.2.** The structure (A, a_1) consisting of facts Pa_1 and Qa_1 (with designated element a_1) has a frontier of size 2 (w.r.t. the class of all finite structures), namely $F = \{(B, a_1), (C, a_1)\}$ where B consists of the facts Pa_1, Pb, Qb and C consists of the facts Qa_1, Pb, Qb, respectively. Note that $((B, a_1), (A, a_1))$ and $((C, a_1), (A, a_1))$ are gap pairs. It can be shown that the structure (A, a_0) has no frontier of size 1 (as such a frontier would have to consist of a structure that contains both facts Pa_1 and Qa_1).

For another example, consider the structure (A', a_1) consisting of facts Pa_1 and Rbb. It is the right hand side of a gap pair (the left hand side being the structure (B', a_1) consisting of the facts Rbb and Pb'), but (A', a_1) has no finite frontier as follows from Theorem 3.7 below.

Frontiers are also closely related to (generalized) homomorphism dualities [16]. We say that a structure $(A, \mathbf{a})$ *has a finite duality* w.r.t. a class $\mathcal{C}$ if there is a finite set of structures

D such that for all $(C, \mathbf{c}) \in \mathcal{C}$, $(A, \mathbf{a}) \to (C, \mathbf{c})$ iff for all $(B, \mathbf{b}) \in D$, $(C, \mathbf{c}) \not\to (B, \mathbf{b})$. If $\mathcal{C}$ is the set of all structures (over the same schema as $(A, \mathbf{a})$), we simply say that $(A, \mathbf{a})$ *has a finite duality.*[2]

▶ **Example 3.3.** In the realm of digraphs, viewed as relational structures without designated elements with a single binary relation, every directed path A of, say, $k > 1$ nodes has finite duality (w.r.t. the class of digraphs). Indeed, it is not difficult to verify that for every digraph C, $A \to C$ iff $C \not\to D$ where D is the digraph with nodes $\{1, \ldots k-1\}$ and edges $\{(i, j) \mid i < j\}$.

▶ **Lemma 3.4.** *Let $\mathcal{C}$ be any class of structures.*

1. *If a structure $(A, \boldsymbol{a})$ has a finite duality w.r.t. $\mathcal{C}$ then $(A, \boldsymbol{a})$ has a finite frontier w.r.t. $\mathcal{C}$.*
2. *If a structure $(A, \boldsymbol{a}) \in \mathcal{C}$ has a finite frontier w.r.t. $\mathcal{C}$ and $\mathcal{C}$ is closed under direct products, then $(A, \boldsymbol{a})$ has a finite duality w.r.t. $\mathcal{C}$.*

Note that the construction of the frontier from the duality is polynomial, while the construction of the duality from the frontier involves an exponential blowup. The following example shows that this is unavoidable.

▶ **Example 3.5.** The path $\circ \xrightarrow{R} \circ \xrightarrow{R_1} \circ \xrightarrow{R} \circ \xrightarrow{R_2} \circ \cdots \circ \xrightarrow{R_n} \circ \xrightarrow{R} \circ$, viewed as a structure without any designated elements, has a frontier (w.r.t. the class of all finite structures) of size polynomial in n, as will follow from Theorem 3.8 below. It is known, however, that any finite duality for this structure must involve a structure whose size is exponential in n, and the example can be modified to use a fixed schema (cf. [31]).

3.1 Frontiers for classes with bounded expansion

The notion of a *class of graphs with bounded expansion* was introduced in [28]. We will not give a precise definition here, but important examples include graphs of bounded degree, graphs of bounded treewidth, planar graphs, and any class of graphs excluding a minor. The same concept of bounded expansion can be applied also to arbitrary structures: a class of structures $\mathcal{C}$ is said to have bounded expansion if the class of Gaifman graphs of structures in $\mathcal{C}$ has bounded expansion. We refer to [29] for more details. Classes of structures of bounded expansion are in many ways computationally well-behaved (cf. for example [23]).

Nešetřil and Ossona de Mendez [27, 29] show that if $\mathcal{C}$ is any class of structures with bounded expansion, then every structure has a finite duality w.r.t. $\mathcal{C}$. It follows by Lemma 3.4 that also every structure has a finite fontier w.r.t. $\mathcal{C}$. Nešetřil and Ossona de Mendez [27, 29] only consider connected structures without designated elements, but their result extends in a straightforward way to the general case of structures with designated elements. Furthermore, it yields an effective procedure for constructing frontiers, although non-elementary (i.e., not bounded by a fixed tower of exponentials).

▶ **Theorem 3.6** (from [27, 29])**.** *Let $\mathcal{C}$ be any class of structures that has bounded expansion. Then every structure $(A, \boldsymbol{a})$ has a finite frontier w.r.t. $\mathcal{C}$, which can be effectively constructed.*

[2] We note here that in the literature on Constraint Satisfaction, it is usual to consider the 'other side' of the duality, i.e, a structure A is said to have finite duality if there exists a finite set of structures F such that for every structure C, $C \to A$ iff for all $B \in F$, $B \not\to C$.

3.2 Polynomial frontiers for c-acyclic structures

Alexe et al. [1], building on Foniok et al. [16], show that a structure has a finite duality if and only if its core is c-acyclic. By Lemma 3.4, this implies that a structure has a finite frontier if and only if its core is c-acyclic.

▶ **Theorem 3.7** (from [16, 1]). *A structure has a finite frontier w.r.t. the class of all structures iff it is homomorphically equivalent to a c-acyclic structure, iff its core is c-acyclic.*

One of our main results is a new proof of the right-to-left direction, which, unlike the original, provides a polynomial-time construction of a frontier from a c-acyclic structure:

▶ **Theorem 3.8.** *Fix a schema $\mathcal{S}$ and $k \geq 0$. Given a c-acyclic structure over $\mathcal{S}$ with k distinguished elements, we can construct in polynomial time a frontier w.r.t. the class of all structures over $\mathcal{S}$ that have k distinguished elements.*

Note that the size of the smallest frontier is in general exponential in k. Indeed, consider the single-element structure $(A, \mathbf{a})$ where A consists of the single fact $P(a)$ and $\mathbf{a} = a, \ldots, a$ has length k. It is not hard to show that every frontier of this (c-acyclic) structure must contain, up to homomorphic equivalence, all structures of the form $(B, \mathbf{b})$ where B consists of two facts, $P(a_1)$ and $P(a_2)$, and $\mathbf{b} \in \{a_1, a_2\}^k$ is a sequence in which both a_1 and a_2 occur. There are exponentially many pairwise homomorphically incomparable such structures.

The proof of Theorem 3.8 is based on a construction that improves over a similar but exponential construction of gap pairs for acyclic structures given in [30, Def. 3.9]. Our results also shed new light on a question posed in the same paper: after presenting a double-exponential construction of duals (for connected structures without designated elements), involving first constructing an exponential-sized gap pair, the authors ask: "*It would be interesting to know to what extent the characterisation of duals can be simplified, and whether the indirect approach via density is optimal.*" This question appeared to have been answered in [31], where a direct method was established for constructing single-exponential size duals. Theorem 3.8 together with Lemma 3.4, however, gives another answer: single-exponential duals can be constructed by going through frontiers (i.e., "via density") as well.

Recall the definition of fg-connectedness from the preliminaries. We first prove a restricted version of Theorem 3.8 for special case of core, fg-connected, c-acyclic structures with the Unique Names Property. We subsequently lift these extra assumptions. A structure $(A, \mathbf{a})$ with $\mathbf{a} = a_1, \ldots a_k$ has the *Unique Names Property (UNP)* if $a_i \neq a_j$ for all $i \leq j$ (cf. [4]).

▶ **Proposition 3.9.** *Given a core, fg-connected, c-acyclic structure with UNP, we can construct in polynomial time (for fixed schema $\mathcal{S}$ and number of distinguished elements k) a frontier w.r.t. the class of all finite structures. Furthermore, the frontier consists of a single structure, which has the UNP.*

Proof. Let a core fg-connected c-acyclic structure $(A, \mathbf{a})$ with UNP be given. Note that each fg-connected structure either (i) consists of a single fact containing only designated elements, or (ii) consists of a number of facts that all contain at least one non-designated element. Therefore, we can distinguish two cases:

Case 1. A consists of a single fact f without non-designated elements. Let $(B, \mathbf{a})$ be the structure whose domain is $\{\mathbf{a}, b\}$, where b is a fresh value distinct from the values in $\mathbf{a}$, and which contains all facts over this domain except f. It is easy to see that $(B, \mathbf{a})$ is a homomorphism dual for $(A, \mathbf{a})$, and consequently, the direct product of the two structures constitutes a singleton frontier for $(A, \mathbf{a})$. Note that this construction is polynomial because we assume that the schema $\mathcal{S}$ and k are both fixed.

Case 2. A consists of one or more facts that each contain a non-designated element. In this case, we construct a singleton frontier $F = \{B\}$ where

- the domain of B consists of
  1. all pairs (a, f) where a is a non-designated element of A and f is a fact of A in which a occurs, and
  2. All pairs (a, id) where a is a designated element of A
- a fact $R((a_1, f_1), \ldots, (a_n, f_n))$ holds in B if and only if $R(a_1, \ldots, a_n)$ holds in A and at least one f_i is a fact that is different from the fact $R(a_1, \ldots, a_n)$ itself.

Note that, in the above construction, id is an arbitrary symbol, used only to simplify notation by ensuring that every element of B can we written as a pair. In what follows we will not distinguish between a designated element a_i and the corresponding pair (a_i, id).

We claim that $F = \{(B, \mathbf{a})\}$ is a frontier for $(A, \mathbf{a})$.

It is clear that the natural projection $h : B \to A$ is a homomorphism.

We claim that there is no homomorphism $h' : A \to B$. Assume, for the sake of a contradiction, that there was such a homomorphism. We may assume that the composition of h and h' is the identity on A (since A is a core, the composition of h and h' is an automorphism of A, that is, an isomorphism from A to itself. By composing h with the inverse of this automorphism if needed, we ensure that its composition with h' is the identity function). In particular, this means that h' maps each designated element a to (a, id) and for each non-designated element a of A, $h'(a) = (a, f)$ for some fact f. For a non-designated element a, let us denote by f_a the unique fact f for which $h'(a) = (a, f_a)$.

We will consider "walks" in A of the form

$$a_1 \xrightarrow{f_{a_1}} a_2 \xrightarrow{f_{a_2}} \ldots a_n$$

with $n \geq 1$, where

1. $a_1, \ldots, a_n$ are non-designated elements,
2. $f_{a_i} \neq f_{a_{i+1}}$, and
3. a_i and a_{i+1} co-occur in fact f_{a_i},

Since A is c-acyclic, the length of any such sequence is bounded by the diameter of A (otherwise some fact would have to be traversed twice in succession, which would violate condition 2). Furthermore, trivially, such a walk of length $n = 1$ exists: just choose as a_1 an arbitrary non-designated element of A. Furthermore, we claim that any such finite sequence can be extended to a longer one: let the fact f_{a_n} be of the form $R(b_1, \ldots, b_m)$ (where $a_n = b_i$ for some $i \leq m$). Since h is a homomorphism, it must map f_{a_n} to some fact $R((b_1, f_{b_1}), \ldots, (b_m, f_{b_m}))$ of B, where some f_{b_j} is a fact that is different from f_{a_n}. We can choose b_j as our element a_{n+1}. Thus, we reach our desired contradiction.

Finally, consider any C with $h : C \to A$ and $A \not\to C$. We construct a function $h' : C \to B$ as follows: consider any element c of C, and let $h(c) = a$. If a is a designated element, we set $h'(c) = (a, \mathsf{id})$. Otherwise, we proceed as follows: since A is c-acyclic and fg-connected, for each non-distinguished element a' of A (other than a itself) there is a unique minimal path in the incidence graph, containing only non-distinguished elements, from a' to a. We can represent this path by a sequence of the form

$$a' = a_0 \xrightarrow{(f_0, i_0, j_0)} a_1 \xrightarrow{(f_1, i_1, j_1)} a_2 \cdots \xrightarrow{(f_{n-1}, i_{n-1}, j_{n-1})} a_n = a$$

where each f_ℓ is a fact of A in which a_ℓ occurs in the i_ℓ-th position and $a_{\ell+1}$ occurs in the j_ℓ-th position. We can partition the non-distinguished elements a' of A (other than a itself) according to the last fact on this path, that is, f_{n-1}. Furthermore, it follows from

fg-connectedness that each fact of A contains a non-distinguished element. It is easy to see that if a fact contains multiple non-distinguished elements (other than a) then they must all belong to the same part of the partition as defined above. Therefore, the above partition on non-distinguished elements naturally extends to a partition on the facts of A. Note that if A contains any facts in which a is the only non-distinguished element, we will refer to these facts as "local facts" and they will be handled separately. In this way, we have essentially decomposed A into a union $A_{\text{local}} \cup \bigcup_i A_i$, where A_{local} contains all local facts and each "component" A_i is a substructure of A consisting of non-local facts, in such a way that (i) different substructures A_i do not share any facts with each other, (ii) different substructures do not share any elements with each other, except for a and distinguished elements, (iii) each A_i contains precisely one fact involving a.

Since we know that $(A, a) \not\to (C, c)$, it follows that either some local fact f of A does not map to C (when sending a to c), or some "component" A_i of A does not map to C through any homomorphism sending a to c. In the first case, we set $h'(a) = (a, f)$. In the second case, we choose such a component (if there are multiple, we choose one of minimal size) and let f be the unique fact in that component containing a (that is, f is the fact f_{n-1} that by construction connected the non-distinguished elements of the component in question to a). We set $h'(c) = (a, f)$. Intuitively, when $h'(c) = (h(c), f)$, then f is a fact of A involving $h(c)$ that "points in a direction where homomorphism from $(A, h(c))$ back to (C, c) fails".

We claim that h' is a homomorphism from C to B: let $R(c_1, \ldots, c_n)$ be a fact of C. Then $R(h(c_1), \ldots, h(c_n))$ holds in A. Let $h'(c_i) = (h(c_i), f_i)$ (where $f_i = \mathsf{id}$ if $h(c_i)$ is a designated element of A). To show that $R(h'(c_1), \ldots, h'(c_n))$ holds in B, it suffices to show that some f_i is different from the fact $R(h(c_1), \ldots, h(c_n))$ itself. If f_i is a local fact, then this follows immediately from the construction. Otherwise, let n_i be the size of the smallest "component" (as defined above) of $(A, h(c_i))$ that does not homomorphically map to (C, c_i), and choose an element c_i with minimal n_i. Then, clearly, f_i must be different from the fact $R(h(c_1), \ldots, h(c_n))$ itself. ◀

Next, we remove the assumptions of fg-connectedness and being a core.

▶ **Proposition 3.10.** *Given a c-acyclic structure with UNP, we can construct in polynomial time a frontier w.r.t. the class of all finite structures. Furthermore, the frontier consists of structures that have the UNP.*

Proof. By Proposition 2.1, we may assume that $(A, \mathbf{a})$ is a core. Note that the c-acyclicity and UNP properties are preserved under the passage from a structure to its core.

Let $(A, \mathbf{a})$ be a structure with designated elements that is UNP and that is a fg-disjoint union of homomorphically incomparable fg-connected structures $(A_1, \mathbf{a}), \ldots, (A_n, \mathbf{a})$. By Proposition 3.9, $(A_1, \mathbf{a}), \ldots, (A_n, \mathbf{a})$ have, respectively, frontiers $F_1, \ldots, F_n$, each consisting of a single structure with the UNP. We may assume without loss of generality that each F_i consists of structures that have the same designated elements $\mathbf{a}$ as A (we know that the structures in question satisfy the UNP, and therefore, modulo homomorphism, we can assume that their designated elements are precisely $\mathbf{a}$).

We claim that $F = \{\big(\biguplus_{j \neq i}(A_j, \mathbf{a})\big) \uplus (B, \mathbf{a}) \mid 1 \leq i \leq n, (B, \mathbf{a}) \in F_i\}$ is a frontier for $(A, \mathbf{a})$ w.r.t. $\mathcal{C}$.

Clearly, each structure in F maps homomorphically to A.

Suppose, for the sake of contradiction, that there were a homomorphism from $h : (A, \mathbf{a}) \to \big(\biguplus_{j \neq i}(A_j, \mathbf{a})\big) \uplus (B, \mathbf{a})$ for some $\mid 1 \leq i \leq n$ and $(B, \mathbf{a}) \in F_i$. Observe that h must send each designated element to itself, and it must send each non-designated element to a non-designated element (otherwise, the composition of h with the backward homomorphism

would be a non-injective endomorphism on $(A, \mathbf{a})$ which would contradict the fact that $(A, \mathbf{a})$ is a core). Since $(A_i, \mathbf{a})$ is fg-connected (and because h cannot send non-designated elements to designated elements), its h-image must be contained either in some $(A_j, \mathbf{a})$ $(j \neq i)$ or in B. The former would cannot happen because A_i and A_j are homomorphically incomparable. The latter cannot happen either, because B belongs to a frontier of A_i.

Finally, let $C \in \mathcal{C}$ be any structure such that there is a homomorphism $h : C \to A$ but $A \not\to C$. Let A_i be a fg-connected component of A such that $A_i \not\to C$. Since A_i is fg-connected, we can partition our structure C as $C_1 \uplus C_2$ where the h-image of C_1 is contained in A_i while the h-image of C_2 is disjoint from A_i. We know that $A_i \not\to C_1$ and therefore $C_1 \to B$ for some $B \in F_i$. Furthermore, we have that $C_2 \to \biguplus_{j \neq i} A_j$. Therefore, $C \to \left(\biguplus_{j \neq i} A_j\right) \uplus B \mid 1 \leq i \leq n, B \mid B \in F_i\}$. ◀

Finally, we can prove Theorem 3.8 itself.

Proof of Theorem 3.8. Let $(A, \mathbf{a})$ be c-acyclic. If it has the UNP, we are done. Consider the other case, where the sequence $\mathbf{a}$ contains repetitions. Let $\mathbf{a}' = a'_1, \ldots, a'_n$ consists of the same elements without repetition (in some order). We construct a frontier for it as follows:

1. Consider the structure $(A, \mathbf{a}')$, which, by construction, has the UNP. Let F be a frontier for $(A, \mathbf{a}')$ (again consisting of structures with the UNP), using Proposition 3.10. Note that, through isomorphism, we may assume that each structure in F has the same designated elements $\mathbf{a}'$. For each $(B, \mathbf{a}') \in F$, we take the structure $(B, \mathbf{a})$.
2. Let k be the length of the tuple $\mathbf{a}$. For each function $f : \{1, \ldots, k\} \to \{1, \ldots, k\}$, whose range has size strictly greater than n, consider structure $(C, \mathbf{c}^f)$ where C contains all facts over the domain $\{1, \ldots, k\}$, and $c^f_i = f(i)$. We take its direct product with $(A, \mathbf{a})$.

It is easy to see that the set of all structures constructed above, constitutes a frontier for $(A, \mathbf{a})$. Indeed, suppose a structure maps to $(A, \mathbf{a})$ but not vice versa. If the tuple of designated elements of the structure in question has the same identity type as the tuple $\mathbf{a}$ (i.e., the same equalities hold between values at different indices in the tuple) then it is easy to see that the structure in question must map to some structure $(B, \mathbf{a})$ as constructed under item 1 above. Otherwise, if the tuple of designated elements of the structure in question does *not* have the same identity type, then it is easy to see that the structure in question must map to $(C^f, \mathbf{c}^f) \times (A, \mathbf{a})$, as constructed under item 2 above, where f reflects the identity type of the designated elements of the structure in question. ◀

As a corollary of Theorem 3.8, we obtain the following interesting by-product:

▶ **Theorem 3.11.** *For a fixed schema $\mathcal{S}$ and $k \geq 0$, the following problem is solvable in NP: given a finite set of structures F and a structure A (all with k designated elements), is F a frontier for A w.r.t. the class of all structures? For some $\mathcal{S}$ and k, it is NP-complete.*

Proof. For the upper bound, we use the fact that, if A is homomorphically equivalent to a c-acyclic structure A', then the core of A is c-acyclic (cf. Theorem 3.7). The problem can therefore be solved in non-deterministic polynomial time as follows:

First we guess a substructure A' and we verify that A' is c-acyclic and homomorphically equivalent to A. Note that the existence of such A' is a necessary precondition for F to be a frontier of A. Furthermore, c-acyclicity can be checked in polynomial time using any PTIME algorithm for graph acyclicity (recall that a structure is c-acyclic if and only if its incidence graph is acyclic after removing all nodes corresponding to designated elements).

Next, we apply Theorem 3.8 to construct a frontier F' for A' (and hence for A). Finally, we verify that each $B \in F$ homomorphically maps to some $B' \in F'$ and, vice versa, every $B' \in F'$

homomorphically maps to some $B \in F$. It is not hard to see that this non-deterministic algorithm has an accepting run if and only if F is a frontier for A.

For the lower bound, we reduce from graph 3-colorability. Let A be the structure, over a 3-element domain, that consists of the facts $R(a,b)$ for all pairs a,b with $a \neq b$. In addition, each of the three elements is named by a constant symbol. Since A is c-acyclic, by Theorem 3.7, it has a frontier F. Now, given any graph G (viewed as a relational structure with binary relation R and without constant symbols), we have that G is 3-colorable if and only if F is a frontier for the disjoint union of A with G. To see that this is the case, note that if G is 3-colorable, then the disjoint union of A with G is homomorphically equivalent to A itself, whereas if G is not 3-colorable, then the disjoint union of A with G is strictly greater than A in the homomorphism order. ◀

3.3 A polynomially frontier-closed class of structures

We call a class $\mathcal{C}$ of structures *frontier-closed* if every structure $(A, \mathbf{a}) \in \mathcal{C}$ has a frontier w.r.t. $\mathcal{C}$, consisting of structures belonging to $\mathcal{C}$. If, moreover, the frontier in question can be constructed from $(A, \mathbf{a})$ in polynomial time, then we say that $\mathcal{C}$ is *polynomially frontier-closed*.

▶ **Theorem 3.12.** *The class of c-connected acyclic structures with 1 designated element is polynomially frontier-closed.* [3]

As will follow from results in Section 4 (cf. Theorems 4.7-4.9 below) the theorem fails if we drop any of the three restrictions in the statement (i.e., c-connectedness, acyclicity, and $k = 1$). However, it might quite well be the case that still holds for some other values of k. Indeed, we conjecture that Theorem 3.12 remains true for any $k > 1$.

4 Unique Characterizations for Conjunctive Queries

In this section, we study the question when a CQ is uniquely characterizable by a finite set of positive and/or negative examples.

▶ **Definition 4.1** (Data Examples, Fitting, Unique Characterizations). *Let $\mathcal{C}$ be a class of k-ary CQs over a schema $\mathcal{S}$ (for some $k \geq 0$), and let q be a k-ary query over $\mathcal{S}$.*

1. *A* data example *is a structure $(A, \boldsymbol{a})$ over schema $\mathcal{S}$ with k distinguished elements. If $\boldsymbol{a} \in q(A)$, we call $(A, \boldsymbol{a})$ a* positive example *(for q), otherwise a* negative example.
2. *Let E^+, E^- be finite sets of data examples. We say that q* fits *(E^+, E^-) if every example in E^+ is a positive example for q and every example in E^- is a negative example for q. We say that (E^+, E^-)* uniquely characterizes *q w.r.t. $\mathcal{C}$ if q fits (E^+, E^-) and every $q' \in \mathcal{C}$ that fits (E^+, E^-) is logically equivalent to q.*

It turns out that there is a precise correspondence between unique characterizations and frontiers. Recall that the canonical structure of a query q is denoted by $\widehat{q}$. Similarly, for any class of CQs $\mathcal{C}$, we will denote by $\widehat{\mathcal{C}}$ the class of structures $\{\widehat{q} \mid q \in \mathcal{C}\}$.

▶ **Proposition 4.2** (Frontiers vs Unique Characterizations). *Fix a schema $\mathcal{S}$ and $k \geq 0$. Let q be any k-ary CQ over $\mathcal{S}$ and $\mathcal{C}$ a class of k-ary CQs over $\mathcal{S}$.*

1. *If F is a frontier for $\widehat{q}$ w.r.t. $\widehat{\mathcal{C}}$, then $(E^+ = \{\widehat{q}\}, E^- = F)$ uniquely characterizes q w.r.t. $\mathcal{C}$.*

[3] Note that for structures with one distinguished element, c-connectedness is the same as connectedness.

2. *Conversely, if* (E^+, E^-) *uniquely characterizes* Q *w.r.t.* $\mathcal{C}$*, then* $F = \{\widehat{q} \times (B, \boldsymbol{b}) \mid (B, \boldsymbol{b}) \in E^-\}$ *is a frontier for* $\widehat{q}$ *w.r.t.* $\widehat{\mathcal{C}}$*.*

Proposition 4.2 allows us to take the results on frontiers from the previous section, and rephrase them in terms of unique characterizations. Incidentally, note that results in [1] imply an analogous relationship between *finite dualities* and uniquely characterizing sets of examples for *unions of conjunctive queries*. We need two more lemmas. Recall that a structure $(A, \mathbf{a})$ corresponds to a conjunctive query only if every distinguished element occurs in at least one fact. Let us call such structures *safe*. The following lemmas, essentially, allow us to ignore unsafe structures, thereby bridging the gap between structures and CQs.

▶ **Lemma 4.3.** *Let* q *be a* k*-ary CQ over schema* $\mathcal{S}$ *and* $\mathcal{C}$ *a class of* k*-ary CQs over* $\mathcal{S}$*. If* q *is uniquely characterized w.r.t.* $\mathcal{C}$ *by positive and negative examples* (E^+, E^-)*, then* q *is uniquely characterized w.r.t.* $\mathcal{C}$ *by* $(\{(A, \boldsymbol{a}) \in E^+ \mid (A, \boldsymbol{a})$ *is safe*$\}, \{(A, \boldsymbol{a}) \in E^- \mid (A, \boldsymbol{a})$ *is safe*$\})$*.*

▶ **Lemma 4.4.** *A safe structure has a finite frontier w.r.t. all structures if and only if it has a finite frontier w.r.t. the class of all safe structures.*

Putting everything together, we obtain the main result of this section. We call a CQ q c-acyclic (or acyclic, or c-connected) if the structure $\widehat{q}$ is c-acyclic (resp. acyclic, c-connected).

▶ **Theorem 4.5.** *Fix a schema and fix* $k \geq 0$*.*

1. *If* $\mathcal{C}$ *is a class of* k*-ary CQs such that* $\widehat{\mathcal{C}}$ *has bounded expansion, then every CQ* $q \in \mathcal{C}$ *is uniquely characterizable w.r.t.* $\mathcal{C}$ *by finitely many positive and negative examples (which can be effectively constructed from the query).*
2. *A* k*-ary CQ* q *is uniquely characterizable by finitely many positive and negative examples (w.r.t. the class of all* k*-ary CQs) iff* q *is logically equivalent to a c-acyclic CQ. Moreover, for a c-acyclic CQ, a uniquely characterizing set of examples can be constructed in polynomial time.*
3. *Assume* $k = 1$ *and let* $\mathcal{C}_{ca}$ *be the class of* k*-ary CQs that are c-connected and acyclic. Then every* $q \in \mathcal{C}_{ca}$ *is uniquely characterizable w.r.t.* $\mathcal{C}_{ca}$ *by finitely many positive and negative examples belonging to* $\widehat{\mathcal{C}_{ca}}$*. Moreover, the set of examples in question can be constructed in polynomial time.*

▶ Remark 4.6. For the purpose of applications discussed in Section 6, we note that Theorem 4.5 remains true if the safety condition for CQs were to be dropped. Indeed, the proof in this case is even simpler, as it does not require Lemma 4.3 and Lemma 4.4.

Theorem 4.5(3) applies to CQs with $k = 1$ that are c-connected, and acyclic. None of these restrictions can be dropped.

▶ **Theorem 4.7.** *The Boolean acyclic connected CQ* $\mathbf{T}() \coloneq Ry_1y_2 \wedge Ry_2y_3 \wedge Ry_3y_4 \wedge Ry_4y_5$ *is not characterized, w.r.t. the class of Boolean acyclic connected CQs, by finitely many acyclic positive and negative examples.*

This shows that in Theorem 4.5(3), the restriction to *unary* queries cannot be dropped. Similarly, the restriction to *c-connected* queries cannot be dropped, and acyclicity cannot be replaced by the weaker condition of c-acyclicity.

▶ **Theorem 4.8.** *The unary acyclic CQ* $\mathbf{T}'(x) \coloneq P(x) \wedge Ry_1y_2 \wedge Ry_2y_3 \wedge Ry_3y_4 \wedge Ry_4y_5$ *is not uniquely characterizable, w.r.t. the class of unary acyclic CQs, by finitely many acyclic positive and negative examples.*

▶ **Theorem 4.9.** *The unary c-acyclic c-connected CQ* $\mathbf{T}''(x) \coloneq Ry_1y_2 \wedge Ry_2y_3 \wedge Ry_3y_4 \wedge Ry_4y_5 \wedge \bigwedge_{i=1\ldots5} Rxy_i$ *is not uniquely characterizable, w.r.t. the class of unary c-acyclic c-connected CQs, by finitely many c-acyclic positive and negative examples.*

5 Exact learnability with membership queries

The unique characterization results in the previous section immediately imply (not-necessarily-efficient) exact learnability results:

▶ **Theorem 5.1.** *Fix a schema and $k \geq 0$. Let $\mathcal{C}$ be a computably enumerable class of k-ary CQs. If $\widehat{\mathcal{C}}$ has bounded expansion, then $\mathcal{C}$ is exactly learnable with membership queries.*

The learning algorithm in question simply enumerates all queries $q \in \mathcal{C}$ and uses membership queries to test if the goal query fits the uniquely characterizing set of examples of q (cf. Theorem 4.5(1)). Unfortunately, this learning algorithm does not run in polynomial time. Indeed, the number of membership queries is not bounded by any fixed tower of exponentials. For the special case of c-acyclic queries, we can do a little better by taking advantage of the fact that a uniquely characterizing set of examples can be constructed in polynomial time. Indeed, the class of c-acyclic k-ary CQs is exponential-time exactly learnable with membership queries: the learner can simply enumerates all c-acyclic queries in order of increasing size. For each query q (starting with the smallest query), it uses Theorem 4.5(2) to test, using polynomially many membership queries, whether the goal query is equivalent to q. After at most $2^{O(n)}$ many attempts (where n is the size of the goal query), the algorithm is guaranteed to find a query that is equivalent to the goal query.[4] Our main result in this section improves on this by establishing *efficient* (i.e., polynomial-time) exact learnability:

▶ **Theorem 5.2.** *For each schema and $k \geq 0$, the class of c-acyclic k-ary CQs is efficiently exactly learnable with membership queries.*

At a high level, the learning algorithm works by maintaining a c-acyclic hypothesis that is an over-approximation of the actual goal query. At each iteration, the hypothesis is strengthened by replacing it with one of the elements of its frontier, a process that is shown to terminate and yield a query that is logically equivalent to the goal query. Note, however, that the frontier of a c-acyclic structure does not, in general, consist of c-acyclic structures. At the heart of the proof of Theorem 5.2 lies a non-trivial argument showing how to turn an arbitrary hypothesis into a c-acyclic one with polynomially many membership queries.

The class of *all* k-ary queries is not exactly learnable with membership queries (even with unbounded amount of time and the ability to ask an unbounded number of oracle queries), because exact learnability with membership queries would imply that every query in the class is uniquely characterizable, which we know is not the case. On the other hand, we have:

▶ **Theorem 5.3** (from [36])**.** *For each schema $\mathcal{S}$ and $k \geq 0$, the class of all k-ary CQs over $\mathcal{S}$ is efficiently exactly learnable with membership and equivalence queries.*

In fact, it follows from results in [36] that the larger class of all *unions of conjunctive queries* is efficiently exactly learnable with membership and equivalence queries (for fixed k and fixed schema). Efficient exact learnability with membership and equivalence queries is not a monotone property of concept classes, but the result from [36] transfers to CQs as well.

▶ Remark 5.4. For the purpose of applications discussed in Section 6, we note that Theorems 5.1- 5.3 remain true if the safety condition for CQs were to be dropped.

[4] Similarly, by Theorem 4.5(3), the class of unary acyclic c-connected queries is exponential-time exactly learnable with subset queries, where a *subset query* is an oracle query asking whether a given CQ from the concept class is implied by the goal query. Subset queries correspond precisely to membership queries where the example is the canonical structure of a query from the concept class.

Related Work. There has been considerable prior work that formally studies the task of identifying some unknown goal query Q from examples. Work in this direction includes learning CQs, Xpath queries, Sparql, tree patterns, description logic concepts, ontologies, and schema mappings among others [8, 36, 18, 33]. We shall describe mostly the previous work regarding learning CQs. Some of the work in this direction ([6, 35, 12, 22, 39] for example) assumes that a background structure A is fixed and known by the algorithm. In this setting, a *example* is a k-ary tuple $(a_1, \ldots, a_k)$ of elements in A, labelled positively or negatively depending on whether it belongs or not to $Q(A)$. In the present paper (as in [36, 19]) we do not fix any background structure (i,e, examples are pairs of the form $(A, \mathbf{a})$). Our setting corresponds also to the extended instances with empty background in [13].

In both cases a number of different learning protocols has been considered. In the reverse-engineering problem (as defined in [38]) it is only required that the algorithm produces a query consistent with the examples. In a similar direction, the problem of determining whether such a query exists has been intensively studied under some variants (satisfiability, query-by-example, definability, inverse satisfiability) [6, 35, 39]. In some scenarios, it is desirable that the query produced by the learner not only explains the examples received during the training phase, but also has also predictive power. In particular, the model considered in [9] follows the paradigm of identification in the limit by Gold and requires that, additionally, there exists a finite set of examples that uniquely determines the target query Q. In a different direction, the model introduced in [18], inspired by the minimum description length principle, requires to produce a hypothesis consistent after some repairs. A third line of work (see [11, 19, 22]) studies this problem under Valiant's probably approximately correct (PAC) model. The present paper is part of a fourth direction based on the exact model of query identification by Angluin. In this model, instead of receiving labelled examples, the learner obtains information about the target query by mean of calls to an oracle. As far as we know, we are the first to study the exact learnability of CQs using a membership oracle.

6 Further Applications

While our main focus in this paper is on unique characterizability and exact learnability for CQs, in this section, we explore some implications for other application domains.

6.1 Characterizability and learnability of LAV schema mappings

A schema mapping is a high-level declarative specifications of the relationships between two database schemas [24]. Two of the most well-studied schema mapping specification languages are *LAV ("Local-as-View")* and *GAV ("Global-as-View")* schema mappings.

In [1], the authors studied the question when a schema mapping can be uniquely characterized by a finite set of data examples. Different types of data examples were introduced and studied, namely positive examples, negative examples, and "universal" examples. In particular, it was shown in [1] that a GAV schema mapping can be uniquely characterized by a finite set of positive and negative examples (or, equivalently, by a finite set of universal examples) if and only if the schema mapping in question is logically equivalent to one that is specified using c-acyclic GAV constraints.

It was shown in [1] that every LAV schema mapping is uniquely characterized by a finite set of universal examples, and that there are LAV schema mappings that are not uniquely characterized by any finite set of positive and negative examples. In this section, we will consider the question *which* LAV schema mappings are uniquely characterizable by a finite set of positive and negative examples, and how to construct such a set of examples efficiently.

We will also consider the exact learnability of LAV schema mappings with membership queries. Exact learnability of GAV schema mappings was studied in [36], where it was shown that GAV schema mappings are learnable with membership and equivalence queries (and, subsequently, also in a variant of the PAC model) but is not exactly learnable with membership queries alone or with equivalence queries alone. The exact learning algorithm for GAV schema mappings from [36] was further put to use and validated experimentally in [37]. Here, we consider exact learnability of LAV schema mappings with membership queries.

▶ **Definition 6.1.** *A* LAV ("Local-As-View") schema mapping *is a triple* $M = (S, T, \Sigma)$ *where* S *and* T *are disjoint schemas (the "source schema" and "target schema"), and* Σ *is a finite set of* LAV constraints*, that is, first-order sentences of the form* $\forall \boldsymbol{x}(\alpha(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y}\phi(\boldsymbol{x}, \boldsymbol{y}))$*, where* $\alpha(\boldsymbol{x})$ *is an atomic formula using a relation from* S*, and* $\phi(\boldsymbol{x}, \boldsymbol{y})$ *is a conjunction of atomic formulas using relations from* T*.*

By a *schema-mapping example* we will mean a pair (I, J) where I is a structure over schema $\mathcal{S}$ without distinguished elements, and J is a structure over schema $\mathcal{T}$ without distinguished elements. We say that (I, J) is a *positive example* for a schema mapping $M = (\mathcal{S}, \mathcal{T}, \Sigma)$ if (I, J), viewed as a single structure over the joint schema $\mathcal{S} \cup \mathcal{T}$, satisfies all constraints in Σ, and we call (I, J) a *negative example* for M otherwise. Note that schema-mapping examples were called *data examples* in [1]. Unique characterizations and learnability with membership queries are defined as before. In particular, by a *membership query*, in the context of learning LAV schema mappings, we will mean an oracle query that consists of a schema-mapping example, which the oracle then labels as positive or negative depending on whether it satisfies the constraints of the goal LAV schema mapping. It is assumed here, that the source and target schemas are fixed and known to the learner.

Given a fixed source schema $\mathcal{S}$, there are only finitely many different possible left-hand sides α for a LAV constraint, up to renaming of variables. Furthermore, if a schema mapping contains two LAV constraints with the same left-hand side, then they can be combined into a single LAV constraint by conjoining the respective right-hand sides. Since the right-hand side of a LAV constraint can be thought of as a CQ, this means that, intuitively, a LAV schema mapping can be thought of as a finite collection of CQs (one for each possible left-hand side). In the light of this observation, it is no surprise that questions about the unique characterizability and learnability of LAV schema mappings can be reduced to questions about the unique characterizability and learnability of CQs.

Let us capture this observation a little more precisely. Let k be the maximum arity of a relation in $\mathcal{S}$, and let $\text{ATOMS}_{\mathcal{S}}$ be the finite set of all atomic formulas using a relation from $\mathcal{S}$ and variables from $\{z_1, \ldots, z_k\}$. Given a LAV schema mapping $M = (\mathcal{S}, \mathcal{T}, \Sigma)$ and an $\alpha(\mathbf{z}) \in \text{ATOMS}_{\mathcal{S}}$, we denote by $q_{M,\alpha}(\mathbf{z})$ the following first-order formula over schema $\mathcal{T}$:

$$\bigwedge_{\substack{\forall \mathbf{x}(\beta(\mathbf{x}) \rightarrow \exists \mathbf{y}\phi(\mathbf{x},\mathbf{y})) \in \Sigma \\ h : \{\mathbf{x}\} \rightarrow \{\mathbf{z}\} \text{ a function s.t. } \beta(h(\mathbf{x})) = \alpha(\mathbf{z})}} \exists \mathbf{y}\phi(h(\mathbf{x}), \mathbf{y})$$

For example, if M consists of the LAV constraints $\forall x_1, x_2, x_3.R(x_1, x_2, x_3) \rightarrow S(x_1, x_2, x_3)$ and $\forall x_1, x_2.R(x_1, x_2, x_2) \rightarrow \exists y T(x_1, y)$, and $\alpha(z_1)$ is $R(z_1, z_1, z_1)$, then $q_{M,\alpha} = S(z_1, z_1, z_1) \wedge \exists y T(z_1, y)$. Similarly, for $\alpha'(z_1, z_2, z_3) = R(z_1, z_2, z_3)$ then $q_{M,\alpha'} = S(z_1, z_2, z_3)$. Note that $q_{M,\alpha}(\mathbf{z})$ can be equivalently written as a not-necessarily-safe CQ over $\mathcal{T}$ (by pulling the existential quantifies to the front).

▶ **Lemma 6.2.** *Let* $M = (\mathcal{S}, \mathcal{T}, \Sigma)$ *be any LAV schema mapping, and let* $\alpha(\boldsymbol{z}) \in \text{ATOMS}_{\mathcal{S}}$ *have* k *many distinct variables. For every structure* $(A, \boldsymbol{a})$*, over schema* $\mathcal{T}$ *and with* k *distinguished elements, the following are equivalent:*

1. $(A, \boldsymbol{a})$ *is a positive data example for* $q_{M,\alpha}(\boldsymbol{z})$*,*
2. *The schema-mapping example* (I, J) *is a positive example for* M*, where* I *is the structure over* $\mathcal{S}$ *consisting of the single fact* $\alpha(\boldsymbol{a})$*, and* $J = A$*.*

We omit the proof, which is straightforward (note that the left-hand side of a LAV constraint can have at most one homomorphism to I, and the latter can be extended to the right-hand side of the constraint to J iff the respective conjunct of $q_{M,\alpha}$ is satisfied).

Intuitively, Lemma 6.2 shows that the behavior of $q_{M,\alpha}$ on arbitrary data examples, is fully determined by the behavior of M on arbitrary schema-mapping examples. The converse turns out to be true as well, that is, the semantics of a LAV schema mapping $M = (\mathcal{S}, \mathcal{T}, \Sigma)$ is determined (up to logical equivalence) by its associated queries $q_{M,\alpha}$ for $\alpha \in \text{ATOMS}_{\mathcal{S}}$:

▶ **Lemma 6.3.** *Two LAV schema mappings* $M_1 = (\mathcal{S}, \mathcal{T}, \Sigma_1)$*,* $M_2 = (\mathcal{S}, \mathcal{T}, \Sigma_1)$ *are logically equivalent iff, for every* $\alpha(\boldsymbol{z}) \in ATOMS_{\mathcal{S}}$*,* $q_{M_1,\alpha}(\boldsymbol{z})$ *and* $q_{M_2,\alpha}(\boldsymbol{z})$ *are logically equivalent.*

Proof. The left-to-right direction follows immediately from the preceding Lemma. For the right-to-left direction: suppose M_1 and M_2 are not logically equivalent. Then they disagree on some schema-mapping example (I, J). Without loss of generality, we may assume that (I, J) is a positive example for M_1 and a negative example for M_2. In particular, one of the LAV constraints in Σ_2 is false in (I, J). Since the left-hand side of a LAV constraint consists of a single atom, it follows that, for some fact $R(\mathbf{a})$ of I, the schema-mapping example $(\{R(\mathbf{a})\}, J)$ is a negative example for M_2. Moreover, an easy monotonicity argument shows that $(\{R(\mathbf{a})\}, J)$ is a positive example for M_1. Let α be obtained from the fact $R(\mathbf{a})$ by replacing each distinct element a_i by a corresponding variable z_i. It follows from Lemma 6.2 that $q_{M_1,\alpha}$ and $q_{M_2,\alpha}$ disagree on the structure $(J, \mathbf{a})$, and are not logically equivalent. ◀

It follows directly from the above Lemmas that the unique characterizability of a LAV schema mapping M reduces to the unique characterizability of each query $q_{M,\alpha}$:

▶ **Lemma 6.4.** *For all LAV schema mappings* $M = (\mathcal{S}, \mathcal{T}, \Sigma)$*, the following are equivalent:*
1. M *is uniquely characterizable by finitely many positive and negative schema-mapping examples (w.r.t. the class of all LAV schema mappings over* $\mathcal{S}, \mathcal{T}$*).*
2. *For each* $\alpha(z_1, \ldots, z_k) \in ATOMS_{\mathcal{S}}$*,* $q_{M,\alpha}(z_1, \ldots, z_k)$ *is uniquely characterizable by finitely many positive and negative data examples w.r.t. the class of all* k*-ary not-necessarily-safe CQs over* $\mathcal{T}$*.*

Intuitively, this shows that a LAV schema mapping is uniquely characterizable iff each of its constraints (joined together according to their left-hand side atom) are. By combining these lemmas with Theorem 4.5 (cf. Remark 4.6), we can link the unique characterizability of a LAV schema mapping to the condition of c-acyclicity. We say that a LAV schema mapping M is c-acyclic if the right-hand side of each of its LAV constraints is a c-acyclic not-necessarily-safe CQ. Note that, in this case, also $q_{M,\alpha}$ is c-acyclic, for each $\alpha \in \text{ATOMS}_{\mathcal{S}}$.

▶ **Theorem 6.5.** *Fix a source schema* $\mathcal{S}$ *and a target schema* $\mathcal{T}$*. A LAV schema mapping* $M = (\mathcal{S}, \mathcal{T}, \Sigma)$ *is uniquely characterizable by a finite set of positive and negative schema-mapping examples if and only if* M *is logically equivalent to a c-acyclic LAV schema mapping. Moreover, if* M *is c-acyclic, then a uniquely characterizing set of positive and negative schema-mapping examples can be constructed in polynomial time (for fixed* $\mathcal{S}, \mathcal{T}$*).*

Proof. The direction going from c-acyclicity to the uniquely characterizing set of schema-mapping examples, follows immediately from the above lemmas together with Theorem 4.5. For the other direction, assume that M is uniquely characterizable by finitely many positive

and negative schema-mapping examples. It follows by Lemma 6.4 that each $q_{M,\alpha}$ is uniquely characterizable by finitely many positive and negative data examples. Hence, each $q_{M,\alpha}$ is logically equivalent to a c-acyclic not-necessarily-safe conjunctive query $q'_{M,\alpha}$. Finally, let $M' = (\mathcal{S}, \mathcal{T}, \Sigma')$, where Σ' consists of all LAV constraints of the form $\forall \mathbf{z}(q_{M,\alpha}(\mathbf{z}) \to \alpha(\mathbf{z}))$ for $\alpha(\mathbf{z}) \in \text{ATOMS}_{\mathcal{S}}$. Then M' is c-acyclic and logically equivalent to M. ◀

Similarly, Lemma 6.2 and Lemma 6.3, together with Theorem 5.2, directly imply:

▶ **Theorem 6.6.** *Fix a source schema $\mathcal{S}$ and a target schema $\mathcal{T}$. The class of c-acyclic LAV schema mappings over $\mathcal{S}, \mathcal{T}$ is efficiently exactly learnable with membership queries.*

Note that the class of *all* LAV schema mappings over $\mathcal{S}, \mathcal{T}$ is *not* exactly learnable with membership queries (assuming that $\mathcal{S}$ is non-empty and $\mathcal{T}$ contains a relation of arity at least 2). This follows immediately from the existence of LAV schema mappings that are not uniquely characterizable by finitely many positive and negative schema-mapping examples.

As mentioned earlier, LAV schema mappings and GAV schema mappings are two of the most well-studied schema mapping languages. GLAV ("Global-and-Local-As-Views") schema mappings is another, which forms a common generalization. An important remaining open question in the area example-driven approaches to schema mapping design is the following [1]: *which GLAV schema mappings are uniquely characterizable by a finite set of examples*?

6.2 Learning description logic concept expressions and ABoxes

Description logics are formal specification languages used to represent domain knowledge. Example-driven and machine-learning based approaches have a long history in this area, and have received renewed interest in the last years [32]. In particular, ontologies specified in the lightweight description logic $\mathcal{ELI}$, focusing on the exact learnability of ontologies using entailment queries and equivalence queries. As we show in this section, our results on c-acyclic CQs have some implications for the exact learnability of $\mathcal{ELI}$ concept expressions.

▶ **Definition 6.7** ($\mathcal{ELI}$ Concept expressions, ABoxes, TBoxes)**.** *Let N_C, N_R, N_I be fixed, disjoint sets, whose members we will refer to as "concept names", "role names", and "individual names", respectively. N_C and N_R are assumed to be finite, while N_I is assumed to be infinite.*

A concept expression *C is an expression built up from from concept names in N_C and $\top$, using conjunction ($C_1 \sqcap C_2$) and existential restriction ($\exists r.C$ or $\exists r^-.C$, where $r \in N_R$).*

An ABox *is a finite set of* ABox axioms *of the form $P(a)$ and/or $r(a, b)$, where $P \in N_C$, $r \in N_R$, and $a, b \in N_I$.*

A TBox *is a finite set of* TBox axioms *$C \sqsubseteq D$, where C, D are concept expressions.*

The semantics of these expressions can be explained by translation to first-order logic:

▶ **Definition 6.8.** *The* correspondence schema *is the schema that contains a unary relation for every $A \in N_C$ and a binary relation for every $r \in N_R$. Through the standard translation from description logic to first-order logic (cf. Table 1), every concept expression C translates to a first-order formula $q_C(x)$ over the correspondence schema. By extension, every TBox $\mathcal{T}$ translates to a finite first-order theory $\mathcal{T}^{fo}$, where $C_1 \sqsubseteq C_2$ translates to $\forall x(q_{C_1}(x) \to q_{C_2}(x))$.*

An ABox can equivalently viewed as a finite structure (without designated elements), whose domain consists of individual names from N_I, and whose facts are the ABox assertions. Since N_I is assumed to be infinite, every finite structure over the correspondence schema can (up to isomorphism) be represented as an ABox. Therefore, in what follows we will use ABoxes and structures interchangeably.

$$
\begin{array}{rcl}
q_P(x) & = & A(x) \text{ for } P \in N_C \\
q_\top(x) & = & \top \\
q_{C_1 \sqcap C_2}(x) & = & q_{C_1}(x) \wedge q_{C_2}(x) \\
q_{\exists r.C}(x) & = & \exists y(r(x,y) \wedge q_C(y)) \\
q_{\exists r^-.C}(x) & = & \exists y(r(y,x) \wedge q_C(y))
\end{array}
$$

Table 1 Standard translation from concept expressions to first-order logic.

	Example	First-order logic translation
ABox:	$\mathcal{A} = \{P(a), r(a,b)\}$	
TBox:	$\mathcal{T} = \{P \sqsubseteq Q \sqcap \exists r.P\}$	$\mathcal{T}^{\text{fo}} = \{\forall x(P(x) \to Q(x) \wedge \exists y(r(x,y) \wedge P(y))\}$
Concept expr:	$C = \exists r.Q$	$q_C(x) = \exists y(r(x,y) \wedge Q(y))$

Table 2 Example description logic ABox, TBox and concept expression.

We can think of an ABox as a (possibly incomplete) list of facts, and a TBox as domain knowledge in the form of rules for deriving more facts. This idea underlies the next definition:

▶ **Definition 6.9.** *A* QA-example *is a pair $(\mathcal{A}, a)$ where $\mathcal{A}$ is an ABox and $a \in N_I$. We say that $(\mathcal{A}, a)$ is a* positive *QA-example for a concept expression C relative to a TBox $\mathcal{T}$ if $a \in certain(C, \mathcal{A}, \mathcal{T})$ where $certain(C, \mathcal{A}, \mathcal{T}) = \bigcap\{q_C(B) \mid \mathcal{A} \subseteq B \text{ and } B \models \mathcal{T}^{fo}\}$. If $a \notin certain(C, \mathcal{A}, \mathcal{T})$, we say that $(\mathcal{A}, a)$ is a negative QA-example for C relative to $\mathcal{T}$.*

The name *QA-example*, here, reflects the fact that the task of computing certain$(C, \mathcal{A}, \mathcal{T})$ is commonly known as *query answering*. It is one of the core inference tasks studied in the description logic literature. In general, there are two variants of the definition of certain$(C, \mathcal{A}, \mathcal{T})$: one where B ranges over finite structures, and one where B ranges over all, finite or infinite, structures. The description logic $\mathcal{ELI}$ that we consider here has been shown to be *finitely controllable* [5], meaning that both definitions are equivalent. For more expressive description logics, this is in general not the case.

▶ **Example 6.10.** Consider the ABox, TBox, and concept expression in Table 2. Every model of $\mathcal{T}^{\text{fo}}$ containing the facts in $\mathcal{A}$ must contain also $r(a,c)$ and $Q(c)$ for some $c \in N_I$. It follows that $a \in$ certain$(C, \mathcal{A}, \mathcal{T})$. In other words, $(\mathcal{A}, a)$ is a positive QA-example for C relative to $\mathcal{T}$. On the other hand, $(\mathcal{A}, b)$ is a negative QA-example for C relative to $\mathcal{T}$.

See [3] for more details on description logic syntax and semantics. We now explain how our results from Section 4 and 5 can be applied here. Although a QA-example is just a data example with one distinguished element, over the correspondence schema, the definition of *positive/negative* QA-examples diverges from the definition of positive/negative data examples, because of the TBox $\mathcal{T}$. For the special case where $\mathcal{T} = \emptyset$, the two coincide:

▶ **Lemma 6.11.** *Let $\mathcal{T} = \emptyset$. A QA-example $(\mathcal{A}, a)$ is a positive (negative) QA-example for a concept expression C relative to $\mathcal{T}$ iff $(\mathcal{A}, a)$ is a positive (negative) data example for $q_C(x)$.*

Lemma 6.11 follows from the well-known monotonicity property of CQs (i.e., whenever $A \subseteq B$, then $q(A) \subseteq q(B)$), which implies that certain$(C, \mathcal{A}, \emptyset) = q_C(\mathcal{A})$.

Concept expressions turn out to correspond precisely to unary, acyclic, c-connected CQs:

▶ **Lemma 6.12.** *The standard translation $q_C(x)$ of every $\mathcal{ELI}$ concept expression C is equivalent to a not-necessarily-safe unary CQ that is acyclic and c-connected. Conversely, every unary, acyclic, c-connected not-necessarily-safe CQ over the correspondence schema is logically equivalent to $q_C(x)$ for some $\mathcal{ELI}$ concept expression C.*

Both directions of Lemma 6.12 can be proved using a straightforward induction.

The above two lemmas, together with Theorem 4.5(2) and Theorem 5.2 (cf. Remark 4.6 and Remark 5.4) immediately yield our main result here. We say that a collection of positive and engative QA-examples *uniquely characterizes* a concept expression C relative to a TBox $\mathcal{T}$ if C fits the examples (relative to $\mathcal{T}$) and every other concept expression that does so is equivalent (relative to $\mathcal{T}$) to C. By a *QA-membership query* we mean an oracle query consisting of a QA example, where the oracle answers yes or no depending on whether the input is a positive QA example or a negative QA example for the goal concept, relative to the TBox. It is assumed that the TBox is fixed and known to the learner.

▶ **Theorem 6.13.** *Let $\mathcal{T} = \emptyset$. Every $\mathcal{ELI}$ concept expression is uniquely characterizable by a finite collection of positive and negative QA examples (relative to $\mathcal{T}$), which can be computed in polynomial time. Furthermore, the class of $\mathcal{ELI}$ concept expressions is efficiently exactly learnable with QA-membership queries.*

Moreover, by Theorem 4.5(3), the uniquely characterizing examples can be constructed so that each example $(\mathcal{A}, a)$ is the canonical QA-example of a concept expression. By the *canonical QA-example* of a concept expression C, here, we mean the QA-example that (viewed as a structure with one distinguished element) is the canonical structure of the not-necessarily-safe CQ $q_C(x)$.

Theorem 6.13 remains true when the concept language is extended with unrestricted existential quantification (of the form $\exists.C$) and a restricted form of the **I-me** self-reference construct introduced in [26], namely where the **I** operator can only occur once, and in the very front of the concept expression. Indeed, it can be shown that this extended concept language (by a straightforward extension of the standard translation) captures precisely the class of c-acyclic unary not-necessarily-safe CQs over the correspondence schema.

▶ **Open Question 6.14.** *Does Theorem 6.13 holds true for arbitrary TBoxes?*

Results in [25] imply that certain$(C, \cdot, \mathcal{T})$ can be expressed in a fragment of monadic Datalog. More precisely, for each $\mathcal{ELI}$ concept expression C and TBox $\mathcal{T}$, there is a monadic Datalog program Π that, such that, for every ABox $\mathcal{A}$, certain$(C, \mathcal{A}, \mathcal{T}) = \Pi(\mathcal{A})$. Moreover, the left-hand side of every Datalog rule of Π is an acyclic, c-connected (unary) CQ. The above question, therefore, may perhaps be approached by studying unique characterizability and exact learnability for a class of acyclic, c-connected monadic Datalog programs. [5]

References

1 Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, December 2011. doi:10.1145/2043652.2043656.

2 Dana Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, April 1988. doi:10.1023/A:1022821128753.

[5] In a recent manuscript [17], a positive answer is given to a weaker variant of the question, namely for the description logic $\mathcal{EL}$, when the learning algorithm is also allowed to ask equivalence queries.

3 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

4 Franz Baader and Werner Nutt. *Basic Description Logics*, page 43–95. Cambridge University Press, USA, 2003.

5 Vince Bárány, Georg Gottlob, and Martin Otto. Querying the guarded fragment. *Logical Methods in Computer Science*, 10(2), 2014. doi:10.2168/LMCS-10(2:3)2014.

6 Pablo Barceló and Miguel Romero. The complexity of reverse engineering problems for conjunctive queries. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 7:1–7:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

7 Google Blog. A reintroduction to our knowledge graph and knowledge panels, 2020. URL: https://blog.google/products/search/about-knowledge-graph-and-knowledge-panels/.

8 Angela Bonifati, Radu Ciucanu, and Aurélien Lemay. Learning path queries on graph databases. In Gustavo Alonso, Floris Geerts, Lucian Popa, Pablo Barceló, Jens Teubner, Martín Ugarte, Jan Van den Bussche, and Jan Paredaens, editors, *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015*, pages 109–120. OpenProceedings.org, 2015.

9 Angela Bonifati, Radu Ciucanu, and Slawek Staworko. Learning join queries from user examples. *ACM Trans. Database Syst.*, 40(4):24:1–24:38, 2016.

10 Ashok K. Chandra Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *ACM Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.

11 William W. Cohen. Cryptographic limitations on learning one-clause logic programs. In Richard Fikes and Wendy G. Lehnert, editors, *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993*, pages 80–85. AAAI Press / The MIT Press, 1993.

12 William W. Cohen. Pac-learning nondeterminate clauses. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, pages 676–681. AAAI Press / The MIT Press, 1994.

13 William W. Cohen. Pac-learning non-recursive prolog clauses. *Artif. Intell.*, 79(1):1–38, 1995.

14 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, July 1983. doi:10.1145/2402.322390.

15 Ronald Fagin, Phokion Kolaitis, Alan Nash, and Lucian Popa. Towards a theory of schema-mapping optimization. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 33–42, January 2008. doi:10.1145/1376916.1376922.

16 Jan Foniok, Jaroslav Nesetril, and Claude Tardif. Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *Eur. J. Comb.*, 29(4):881–899, 2008.

17 Maurice Funk, Jean Christoph Jung, and Carsten Lutz. Actively learning el-concepts and queries in the presence of an ontology. Manuscript, 2020.

18 Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. *J. ACM*, 57(2):6:1–6:37, 2010. doi:10.1145/1667053.1667055.

19 David Haussler. Learning conjunctive concepts in structural domains. *Mach. Learn.*, 4:7–40, 1989.

20 Pavol Hell and Jaroslav Nešetřil. The Core of a Graph. *Discrete Mathematics*, 109:117–126, 1992.

21 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford lecture series in mathematics and its applications*. Oxford University Press, 2004.

22 Kouichi Hirata. On the hardness of learning acyclic conjunctive queries. In Hiroki Arimura, Sanjay Jain, and Arun Sharma, editors, *Algorithmic Learning Theory, 11th International*

Conference, ALT 2000, Sydney, Australia, December 11-13, 2000, Proceedings, volume 1968 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.

23 Wojtek Kazana and Luc Segoufin. First-order queries on classes of structures with bounded expansion. *Logical Methods in Computer Science*, Volume 16, Issue 1, February 2020. URL: https://lmcs.episciences.org/6156.

24 Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the Twenty-Fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '05, page 61–75, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1065167.1065176.

25 Carsten Lutz and Frank Wolter. The data complexity of description logic ontologies. *Logical Methods in Computer Science*, 13, November 2016. doi:10.23638/LMCS-13(4:7)2017.

26 Maarten Marx. Narcissists, stepmothers and spies. In Ian Horrocks and Sergio Tessaris, editors, *Proceedings of the 2002 International Workshop on Description Logics (DL2002), Toulouse, France, April 19-21, 2002*, volume 53 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2002. URL: http://ceur-ws.org/Vol-53/narcists.pdf.

27 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion iii. restricted graph homomorphism dualities. *European Journal of Combinatorics*, 29(4):1012–1024, 2008. Homomorphisms: Structure and Highlights. doi:10.1016/j.ejc.2007.11.019.

28 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion i. decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008. doi:10.1016/j.ejc.2006.07.013.

29 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity (Graphs, Structures, and Algorithms)*, volume 28. Springer, January 2012. doi:10.1007/978-3-642-27875-4.

30 Jaroslav Nešetřil and Claude Tardif. Duality theorems for finite structures (characterising gaps and good characterisations). *Journal of Combinatorial Theory, Series B*, 80(1):80–97, 2000. doi:10.1006/jctb.2000.1970.

31 Jaroslav Nešetřil and Claude Tardif. Short answers to exponentially long questions: Extremal aspects of homomorphism duality. *SIAM J. Discret. Math.*, 19(4):914–920, August 2005. doi:10.1137/S0895480104445630.

32 Ana Ozaki. Learning description logic ontologies: Five approaches. where do they stand? *KI - Künstliche Intelligenz*, April 2020. doi:10.1007/s13218-020-00656-9.

33 Slawek Staworko and Piotr Wieczorek. Learning twig and path queries. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 140–154. ACM, 2012.

34 Balder ten Cate, Laura Chiticariu, Phokion Kolaitis, and Wang-Chiew Tan. Laconic schema mappings: Computing the core with sql queries. *Proc. VLDB Endow.*, 2(1):1006–1017, August 2009. doi:10.14778/1687627.1687741.

35 Balder ten Cate and Víctor Dalmau. The product homomorphism problem and applications. In Marcelo Arenas and Martín Ugarte, editors, *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, volume 31 of *LIPIcs*, pages 161–176. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

36 Balder ten Cate, Víctor Dalmau, and Phokion G. Kolaitis. Learning schema mappings. *ACM Trans. Database Syst.*, 38(4):28:1–28:31, 2013. doi:10.1145/2539032.2539035.

37 Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang-Chiew Tan. Active learning of GAV schema mappings. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 355–368. ACM, 2018. doi:10.1145/3196959.3196974.

38 Yaacov Y. Weiss and Sara Cohen. Reverse engineering spj-queries from examples. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 151–166. ACM, 2017.

39 Ross Willard. Testing expressibility is hard. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 2010.

The Complexity of Aggregates over Extractions by Regular Expressions

Johannes Doleschal
Universität Bayreuth, Germany
Hasselt University, Belgium

Noa Bratman
Technion - Israel Institute of Technology, Haifa, Israel

Benny Kimelfeld
Technion - Israel Institute of Technology, Haifa, Israel

Wim Martens
Universität Bayreuth, Germany

Abstract

Regular expressions with capture variables, also known as "regex-formulas," extract relations of spans (intervals identified by their start and end indices) from text. In turn, the class of regular document spanners is the closure of the regex formulas under the Relational Algebra. We investigate the computational complexity of querying text by aggregate functions, such as sum, average, and quantile, on top of regular document spanners. To this end, we formally define aggregate functions over regular document spanners and analyze the computational complexity of exact and approximate computation. More precisely, we show that in a restricted case, all studied aggregate functions can be computed in polynomial time. In general, however, even though exact computation is intractable, some aggregates can still be approximated with fully polynomial-time randomized approximation schemes (FPRAS).

2012 ACM Subject Classification Information systems → Information extraction; Theory of computation → Database query languages (principles); Theory of computation → Problems, reductions and completeness

Keywords and phrases Information extraction, document spanners, regular expressions, aggregation functions

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.10

Funding This work was supported by the German-Israeli Foundation for Scientific Research and Development (GIF), grant I-1502-407.6/2019. The work of Benny Kimelfeld was also supported by the Israel Science Foundation (ISF), Grant 768/19, and the German Research Foundation (DFG) Project 412400621 (DIP program).

1 Introduction

Information extraction commonly refers to the task of extracting structured information from text. A document spanner (or just spanner for short) is an abstraction of an information extraction program: it states how to transform a document into a relation over its spans. More formally, a *document* is a string d over a finite alphabet, a *span* of d represents a substring of d by its start and end positions, and a *spanner* is a function that maps every document d into a relation over the spans of d [7]. The spanner framework has originally been introduced as the theoretical basis underlying IBM's SQL-like rule system for information extraction, namely SystemT [15, 18]. The most studied spanner instantiation is the class of *regular spanners* – the closure of *regex-formulas* (regular expressions with capture variables) under the standard operations of the relational algebra (projection, natural join, union,

and difference). Equivalently, the regular spanners are the ones expressible as *variable-set automata* (VSet-automata for short) – nondeterministic finite-state automata that can open and close capture variables. These spanners extract from the text relations wherein the capture variables are the attributes.

While regular spanners and natural generalizations thereof are the basis of rule-based systems for text analytics, they are also used implicitly in other types of systems, and particularly ones based on statistical models and machine learning. Rules similar to regular spanners are used for *feature generators* of graphical models (e.g., Conditional Random Fields) [17, 32], *weak constraints* of Markov Logic Networks [28] and extensions such as DeepDive [31], and the generators of *noisy training data* ("labeling functions") in the state-of-the-art Snorkel system [29]. Further connections to regular spanners can potentially arise from efforts to express artificial neural networks for natural language processing as finite-state automata [21, 22, 34]. The computational complexity of evaluating regular spanners has been well studied from various angles, including the data and combined complexity of answer enumeration [1, 8, 10, 20], the cost of combining spanners via relational algebra operators [26] and recursive programs [27], their dynamic complexity [11], evaluation in the presence of weighted transitions [5], and the ability to distribute their evaluation over fragments of the document [4].

In this paper, we study the computational complexity of evaluating *aggregate functions* over regular spanners. These are queries that map a document d and a spanner P into a number $\alpha(P(d))$, where $P(d)$ is the relation obtained by applying P to d and α is a standard aggregate function: count, sum, average, min, max, or quantile. There are various scenarios where queries that involve aggregate functions over spanners can be useful. For example, such queries arise in the extraction of statistics from textual resources like medical publications [25] and news reports [30]. As another example, when applying advanced text search or protein/DNA motif matching using regular expressions [3, 24], the search engine typically provides the (exact or approximate) number of answers, and we would like to be able to compute this number without actually computing the answers, especially when the number of answers is prohibitively large. Finally, when programming feature generators or labeling functions in extractor development, the programmer is likely to be interested in aggregate statistics and summaries for the extractions (e.g., to get a holistic view of what is being extracted from the dataset, such as quantiles over extracted ages and so on), and again, we would like to be able to estimate these statistics faster than it takes to materialize the entire set of answers.

Our main objective in this work is to understand when it is tractable to compute $\alpha(P(d))$. This question raises closely related questions that we also discuss in the paper, such as when the materialization of intermediate results (which can be exponentially large) can be avoided. Furthermore, when the exact computation of $\alpha(P(d))$ is intractable, we study whether it can be approximated.

At the technical level, each aggregate function (with the exception of count) requires a specification of how an extracted tuple of spans represents a number. For example, the number 21 can be represented by the span of the string "21", "21.0", "twenty one", "twenty first", "three packs of seven" and so on. To abstract away from specific textual representations of numbers, we consider several means of assigning weights to tuples. To this end, we assume that a (representation of a) *weight function* w, which maps every tuple of $P(d)$ into a number, is part of the input of the aggregate functions. Hence, the general form of the aggregate query we study is $\alpha(P, d, w)$. The direct approach to evaluating $\alpha(P, d, w)$ is to compute $P(d)$, apply w to each tuple, and apply α to the resulting sequence of numbers. This approach

works well if the number of tuples in $P(d)$ is manageable (e.g., bounded by some polynomial). However, the number of tuples in $P(d)$ can be exponential in the number of variables of P, and so, the direct approach takes exponential time in the worst case. We will identify several cases in which $P(d)$ is exponential, yet $\alpha(P(d))$ can be computed in polynomial time.

It is not very surprising that, at the level of generality we adopt, each of the aggregate functions is intractable (#P-hard) in general. Hence, we focus on several assumptions that can potentially reduce the inherent hardness of evaluation:

- Restricting to positive numbers;
- Restricting to weight functions w that are determined by a single span or defined by (unambiguous) weighted VSet-automata;
- Restricting to spanners that are represented by an unambiguous variant of VSet-automata;
- Allowing for a randomized approximation (FPRAS, i.e., fully polynomial randomized approximation schemes).

Our analysis shows which of these assumptions bring the complexity down to polynomial time, and which is insufficient for tractability. Importantly, we derive an interesting and general tractable case for each of the aggregate functions we study.

The problem of counting the number of extractions of a VSet-automaton has been studied by Florenzano et al. [8]. The approximate version has been studied by Arenas et al. [2] who give a polynomial-time algorithm for uniformly sampling from the space of accepted words of a given length for an NFA, and to estimate the number of such accepted words. Using that sampling, they establish an FPRAS for counting the number of tuples extracted by a VSet-automaton (i.e., the Count aggregate function). Our FPRAS results are also based on their results. The counting problem is also implicitly discussed by Doleschal et al. [5] who study annotation by semiring elements over weighted VSet-automata. Throughout the paper, we explain the connection between all of these and our work in more detail. Yet, to the best of our knowledge, this paper is the first to consider aggregate functions over numerical values extracted by document spanners.

The remainder of the paper is organized as follows. In Section 2, we give preliminary definitions and notation. In Section 3, we give a general summary of the main results of the paper; we expand on these results in the later sections. In Sections 4, 5 and 6 we describe our investigation for single-variable weight functions, polynomial-time weight functions and regular weight functions, respectively. Finally, we consider approximate evaluations in Section 7 and conclude in Section 8. Due to space constraints, we sometimes omit proofs or only provide a proof sketch.

2 Preliminaries

The cardinality of a set A is denoted by $|A|$. A *multiset* over A is a function $M : A \to \mathbb{N}$. We call $M(a)$ the *multiplicity* of a in M and say that $a \in M$ if $M(a) > 0$. The *size* of M denoted $|M|$, is the sum $\sum_{a \in A} M(a)$, which may be infinite. We denote multisets in brackets $\{\!\!\{$ and $\}\!\!\}$ in the usual way. E.g., in $M = \{\!\!\{1, 1, 3\}\!\!\}$ we have that $M(1) = 2$ and $M(3) = 1$.

We revisit some definitions from the *document spanners framework* [7]. Let Σ be a finite set of symbols called the *alphabet*. By Σ^* we denote the set of all finite words over Σ, also called *documents*. The *length* $|d|$ of document $d = \sigma_1 \cdots \sigma_n \in \Sigma^*$ (with every $\sigma_i \in \Sigma$) is n. A *span* of d is an expression of the form $[i, j\rangle$ with $1 \leq i \leq j \leq n + 1$. For a span $[i, j\rangle$ of d, we denote by $d_{[i,j\rangle}$ the word $\sigma_i \cdots \sigma_{j-1}$. For a document d, we denote by *Spans(d)* the set of all possible spans of d. Two spans $[i_1, j_1\rangle$ and $[i_2, j_2\rangle$ are *equal* if $i_1 = i_2$ and $j_1 = j_2$.

The framework focuses on functions that extract spans from documents and assigns them to variables. To this end, we fix a countably infinite set Var of *span variables*, which range over spans, such that Var and Σ are disjoint. A *d-tuple* t is a total function from a finite set of span variables into *Spans(d)*. We denote the domain of t by Vars(t). If the document d is clear from the context, we sometimes say *tuple* instead of d-tuple. A set of d-tuples over the same variables is called a *d-relation*. For a d-tuple t and a set $Y \subseteq \text{Vars(t)}$ we define the d-tuple $\text{t}{\restriction}Y$ as the restriction of t to the variables in Y. A *document spanner* is a function P that maps every document d into a finite d-relation, which we denote by $P(d)$. By $\text{Vars}(P)$ we denote the domain of the tuples in $P(d)$, which we call the variables of the spanner. We refer to Appendix A for the definition of algebraic operations on spanners.

Variable Set-Automata. This paper will focus on *regular spanners*, which can be defined as follows. A *variable-set automaton (VSet-automaton)* is an NFA that accepts words with *variable operations*, which are symbols of the form "$x\vdash$" (open x) and "$\dashv x$" (close x), where x is a variable. More precisely, for a set of variables $V \subseteq \mathsf{Var}$, the set of *variable operations over* V is $\Gamma_V := \{x\vdash, \dashv x \mid x \in V\}$, which we assume to be disjoint from Σ and Var.

A VSet-automaton is a tuple $A := (\Sigma, V, Q, q_0, Q_F, \delta)$, where Σ is a finite set of alphabet symbols, $V \subseteq \text{Vars}$ is a finite set of variables, Q is a finite set of states, $q_0 \in Q$ is a start state, $Q_F \subseteq Q$ is a set of final states, and $\delta : Q \times (\Sigma \cup \Gamma_V \cup \{\varepsilon\}) \to 2^Q$ is the transition function. We refer to words over the alphabet $\Sigma \cup \Gamma_V$ as *ref-words* [9] and, therefore, the *ref-word language* $\mathcal{R}(A)$ of A is the set of words accepted by the NFA $(\Sigma \cup \Gamma_V, Q, q_0, Q_F, \delta)$, which is an ordinary NFA over alphabet $(\Sigma \cup \Gamma_V)$.

We now discuss how A defines a spanner. The set $\text{Vars}(\mathbf{r})$ is the set of variables x such that $x\vdash$ or $\dashv x$ occur in ref-word $\mathbf{r}$. A ref-word $\mathbf{r}$ is *valid* if, for each $x \in \text{Vars}(\mathbf{r})$, it has precisely one occurrence of $x\vdash$ and precisely one occurrence of $\dashv x$, which is after the occurrence of $x\vdash$. It is *valid for* V if it is valid and if $V = \text{Vars}(\mathbf{r})$. By $\mathcal{VR}(A)$ we denote the words in $\mathcal{R}(A)$ that are valid for V.

Consider the mapping $\text{clr} : (\Sigma \cup \Gamma_{\text{Vars}})^* \to \Sigma^*$ (pronounced "clear"), as $\text{clr}(\sigma) := \sigma$ for every $\sigma \in \Sigma$ and $\text{clr}(\sigma) := \varepsilon$ for every $\sigma \in \Gamma_{\text{Vars}}$, where ε denotes the empty word. If $\mathbf{r}$ is a valid ref-word with $\text{clr}(\mathbf{r}) = d$, with $\mathbf{r} = \mathbf{r}_x^{\text{pre}} \cdot x\vdash \cdot \mathbf{r}_x \cdot \dashv x \cdot \mathbf{r}_x^{\text{post}}$, then the *d-tuple* $\text{tup}_{\mathbf{r}}$ *induced by* $\mathbf{r}$ is defined as $\text{tup}_{\mathbf{r}}(x) := [i, j\rangle$, where $i := |\text{clr}(\mathbf{r}_x^{\text{pre}})| + 1$ and $j := i + |\text{clr}(\mathbf{r}_x)|$ for every variable $x \in \text{Vars}(\mathbf{r})$. The *spanner* $[\![A]\!]$ induced by A maps each document d to $\{\text{tup}_{\mathbf{r}} \mid \mathbf{r} \in \mathcal{VR}(A), \text{clr}(\mathbf{r}) = d\}$.

Notice that only the valid ref-words of A produce output tuples. A VSet-automaton is *functional* if it only accepts valid ref-words, i.e., $\mathcal{VR}(A) = \mathcal{R}(A)$. Since VSet-automata can always be translated into equivalent functional VSet-automata [9, Proposition 3.9], we assume in this paper that VSet-automata are functional. This is a common assumption for document spanners involving regular languages [7, 9, 26]. In the following, we denote by VSA the class of functional VSet-automata.

Regular spanners can also be represented by *regex-formulas*, which are regular expressions that may include variables. We refer to Appendix B for a formal definition.

Aggregate Queries. Aggregation functions, such as min, max, and sum operate on numerical values from database tuples, whereas all the values of d-tuples are spans. Yet, these spans may represent numerical values, from the document d, encoded by the captured words (e.g., "3," "three," "March" and so on). To connect spans to numerical values, we will use *weight functions* w that map document/tuple pairs to numbers in $\mathbb{Q}$, that is, if d is a document and t is a d-tuple then $w(d, \text{t}) \in \mathbb{Q}$. We discuss weight functions in more detail in Section 3.3.

$d_{x_{\text{loc}}}$	$d_{x_{\text{events}}}$	$w(d,\mathrm{t})$
Belgium	7	7
France	10-15	10
Luxembourg	4	4
Berlin	three	3

x_{loc}	x_{events}
$[23, 30\rangle$	$[11, 12\rangle$
$[41, 47\rangle$	$[32, 37\rangle$
$[54, 64\rangle$	$[49, 50\rangle$
$[75, 81\rangle$	$[66, 71\rangle$

Figure 1 A document d (top), a relation with corresponding weights (bottom left), and the corresponding d-relation R (bottom right).

▶ **Example 2.1.** Consider the document in Figure 1 and assume that we want to calculate the total number of mentioned events. The table at the bottom left depicts a possible extraction of locations with their number of events, where each tuple is annotated with a weight $w(d,\mathrm{t})$. The table on the bottom right depicts the corresponding span relation. To get an understanding of the total number of events, we may want to take the sum over the weights of the extracted tuples, namely $7 + 10 + 4 + 3 = 24$.

For a spanner P, a document d, and weight function w, we denote by $Img(P, d, w)$ the set of weights of output tuples of P on d, that is, $Img(P, d, w) = \{w(d,\mathrm{t}) \mid \mathrm{t} \in P(d)\}$. Furthermore, let $Img(w) \subseteq \mathbb{Q}$ be the set of weights assigned by w, that is, $k \in Img(w)$ if and only if there is a document d and a d-tuple t with $w(d,\mathrm{t}) = k$.

▶ **Definition 2.2.** *Let d be a document and A be a VSet-automaton such that $[\![A]\!](d) \neq \emptyset$. Let $P = [\![A]\!]$, let w be a weight function, and $q \in \mathbb{Q}$ with $0 \leq q \leq 1$. We define the following spanner aggregation functions:*

$$\mathrm{Count}(P,d) := |P(d)|$$

$$\mathrm{Min}(P,d,w) := \min_{\mathrm{t}\in P(d)} w(d,\mathrm{t}) \qquad \mathrm{Max}(P,d,w) := \max_{\mathrm{t}\in P(d)} w(d,\mathrm{t})$$

$$\mathrm{Sum}(P,d,w) := \sum_{\mathrm{t}\in P(d)} w(d,\mathrm{t}) \qquad \mathrm{Avg}(P,d,w) := \frac{\mathrm{Sum}(P,d,w)}{\mathrm{Count}(P,d)}$$

$$q\text{-Quantile}(P,d,w) := \min\left\{ r \in Img(P,d,w) \;\middle|\; \frac{|\{\mathrm{t} \in P(d) \mid w(d,\mathrm{t}) \leq r\}|}{|P(d)|} \geq q \right\}$$

Observe that $0\text{-Quantile}(P,d,w) = \mathrm{Min}(P,d,w)$ *and* $1\text{-Quantile}(P,d,w) = \mathrm{Max}(P,d,w)$.

Main Problems. Let $\mathcal{P}$ be a class of regular document spanners and $\mathcal{W}$ be a class of weight functions. We define the following problems.

Count$[\mathcal{P}]$	Sum$[\mathcal{P},\mathcal{W}]$
Input: Spanner $P \in \mathcal{P}$ and document $d \in \Sigma^*$.	Input: Spanner $P \in \mathcal{P}$, document $d \in \Sigma^*$, a weight function $w \in \mathcal{W}$.
Task: Compute $\mathrm{Count}(P,d)$.	Task: Compute $\mathrm{Sum}(P,d,w)$.

The problems Average$[\mathcal{P},\mathcal{W}]$, q-Quantile$[\mathcal{P},\mathcal{W}]$, Min$[\mathcal{P},\mathcal{W}]$, and Max$[\mathcal{P},\mathcal{W}]$ are defined analogously to Sum$[\mathcal{P},\mathcal{W}]$. Notice that all these problems study *combined complexity*. Since the number of tuples in $P(d)$ is always in $O(|d|^{2k})$, where k is the number of variables of the

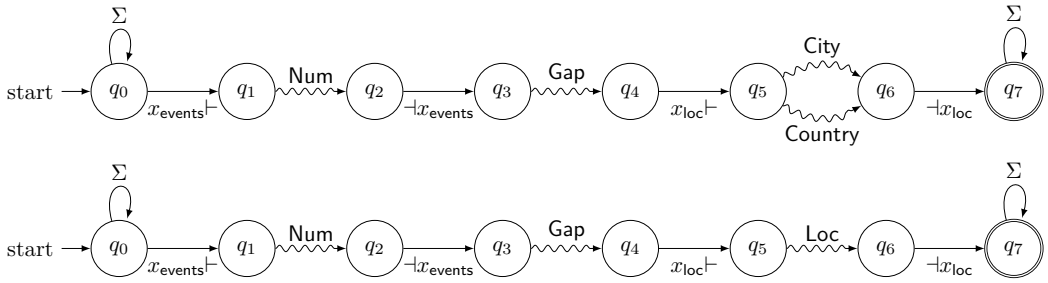


Figure 2 Two example VSet-automata that extract the d-relation R on input d as defined in Figure 1. For the sake of presentation, the automata are simplified as follows: Num is a sub-automaton matching anything representing a number (of events) or range, Gap is a sub-automaton matching sequences of at most three words, City and Country are sub-automata matching city and country names respectively. Loc is a sub-automaton for the union of City and Country. All these sub-automata are assumed to be deterministic.

spanner P, the *data complexity* of all the problems is in FP: one can just materialize $P(d)$ and apply the necessary aggregate. Under combined complexity, we will therefore need to find ways to avoid materializing $P(d)$ to achieve tractability.

3 Main Results

In this section we present our main results. We present the results for spanners represented as VSet-automata, but they also hold for their regular expression counterpart. Notice that this is not trivial because translating finite automata to regular expressions can incur an unavoidable exponential blow-up [6].

Unambiguous VSet-Automata. For a number of our tractability results, it is important that the VSet-automata in the input are *unambiguous*. In order to define unambiguity, we fix a total, linear order $\prec$ on the set Γ_{Var} of variable operations, such that $x\vdash \prec \dashv x$ for every variable x. A VSet-automaton $A = (\Sigma, \text{Vars}, Q, q_0, Q_f, \delta)$ satisfies the *variable order condition* if $v \prec v'$ for every $v, v' \in \Gamma_{\text{Vars}}$ for which there are $q_1, q_2, q_3 \in Q$ such that $q_2 \in \delta(q_1, v)$ and $q_3 \in \delta(q_2, v')$. The variable order condition ensures that, for every document $d \in \Sigma^*$ and every tuple $\mathrm{t} \in [\![A]\!](d)$, there is exactly one ref-word $\mathbf{r} \in \mathcal{VR}(A)$ with $\text{tup}_{\mathbf{r}} = \mathrm{t}$.

A *run* of A on $\mathbf{r} = \sigma_1 \cdots \sigma_n$ is a sequence $q_0 \cdots q_n$ of states of A such that $q_i \in \delta(q_{i-1}, \sigma_i)$ for every $i \in \{1, \ldots, n\}$. It is *accepting* if $q_n \in Q_f$. A VSet-automaton A is *unambiguous*, if

1. A satisfies the variable order condition and
2. there is exactly one accepting run of A on every $\mathbf{r} \in \mathcal{R}(A)$.

In the following, we denote by uVSA the class of unambiguous VSet-automata.

▶ **Example 3.1.** The d-relation on the bottom right of Figure 1 can be extracted from d by a spanner that matches textual representations of numbers (or ranges) in the variable x_{events}, followed by a city or country name, matched in x_{loc}. Figure 2 shows how two such VSet-automata may look like. Note that some strings, like Luxembourg are the name of a city as well as a country. Thus, the upper automaton is ambiguous, because the tuple with Luxembourg is captured twice. The lower automaton is unambiguous, because the sub-automaton for Loc only matches such names once.

Table 1 Known results on the complexity of COUNT.

Aggregate	Spanner	Complexity	Reference
Count	uVSA	in FP	[2, Corollary 4.2]
Count	VSA	#P-complete	[8, Theorem 5.2] (implicit)
Count	VSA	FPRAS	[2, Corollary 4.1]

Complexity Classes. We assume familiarity with the complexity classes FP (polynomial-time computable functions), #P, $\text{FP}^{\#\text{P}}$, and OptP. (We provide some background in Appendix C.) Unless mentioned otherwise, we use Cook reductions, also known as Turing reductions. Notice that, under Turing reductions, #P-complete problems are also $\text{FP}^{\#\text{P}}$-complete.

3.1 Known Results

A number of results on COUNT are already known or easily follow from known results, see Table 1 and Theorem 3.2. Two observations can be made from this table. First, COUNT requires the input spanner to be *unambiguous* for tractability. This tractability implies that COUNT can be computed without materializing the possibly exponentially large set $P(d)$ if the spanner is unambiguous. Second, if the spanner is not unambiguous then, due to #P-completeness of COUNT, we do not know an efficient algorithm for its exact computation (and therefore may have to materialize $P(d)$), but COUNT can be *approximated* by an FPRAS. We will explore to which extent this picture generalizes to other aggregates.

▶ **Theorem 3.2** (Arenas et al. [2], Florenzano et al. [8]). *COUNT*[uVSA] *is in* FP *and* *COUNT*[VSA] *is* #P*-complete. Furthermore,* *COUNT*[VSA] *can be approximated by an* FPRAS.

3.2 Overview of New Results

Our new complexity results are summarized in Table 2. By now the reader is familiar with the aggregate problems and the types of spanners we study. In the next subsection, we will define the different representations of weight functions that we will use. Here, SINGLE are single-variable weight functions, POLY are polynomial-time computable weight functions, and REG (resp., UREG) are weight functions represented by weighted (resp., unambiguous weighted) VSet-automata. We use the following notation in the table.

▶ **Notation 3.3.** *If $\mathcal{W}$ is a class of representations of weight functions and S is a set, we denote by $\mathcal{W}_S$ the subset of $\mathcal{W}$ that represent a weight function w with $Img(w) \subseteq S$. Typical sets that we use for S are $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}^+ := \{q \in \mathbb{Q} \mid q \geq 0\}$, and $\mathbb{B} := \{0, 1\}$.*

Entries in the table should be read from left to right. For instance, the FP result in the first row states that the problems MIN, for both spanner classes uVSA and VSA, and for all three classes SINGLE, UREG and REG of weight functions is in FP. Likewise, the second row states that the same problems with $\text{POLY}_{\mathbb{N}}$ weight functions become OptP-complete and that the existence of an FPRAS would imply a collapse of the polynomial hierarchy.

In general, the table gives a detailed overview of the impact of (1) unambiguity of spanners and (2) different weight function representations on the complexity of computing aggregates.

Table 2 Detailed overview of complexities of aggregate problems for document spanners. All these results are new. By X-c we denote that the problem is complete for class X. The "no FPRAS" claims assume that the polynomial hierarchy does not collapse to the second level.

Aggregate	Spanner	Weights	Complexity
Min	uVSA, VSA	Single, UReg, Reg	in FP
		$\text{Poly}_{\mathbb{N}}$	OptP-c, no FPRAS
Max	uVSA, VSA	Single, UReg	in FP
		$\text{Reg}_{\mathbb{N}}$, $\text{Poly}_{\mathbb{N}}$	OptP-c, no FPRAS
Sum, Average	uVSA	Single, UReg	in FP
		Reg, $\text{Poly}_{\mathbb{Z}}$	$\text{FP}^{\#\text{P}}$-c, no FPRAS
	VSA	$\text{Single}_{\mathbb{Q}^+}$	$\text{FP}^{\#\text{P}}$-c, FPRAS
		Single, UReg, Reg, $\text{Poly}_{\mathbb{Z}}$	$\text{FP}^{\#\text{P}}$-c, no FPRAS
q-Quantile	uVSA	Single	in FP
		UReg, Reg, $\text{Poly}_{\mathbb{Z}}$	$\text{FP}^{\#\text{P}}$-c, no FPRAS
	VSA	Single, UReg, Reg, $\text{Poly}_{\mathbb{Z}}$	$\text{FP}^{\#\text{P}}$-c, no FPRAS
q-Quantile (position)	VSA	Poly	Polynomial time positional approximation

3.3 Results for Different Weight Functions

We formalize how we represent the weight functions for our new results. Recall that weight functions w map pairs consisting of a document d and d-tuple t to values in $\mathbb{Q}$.

3.3.1 Single-Variable Weight Functions

The simplest type of weight functions we consider are the *single-variable* weight functions. To facilitate presentation, we assume that a designated variable x is always present in t. A *single-variable (SV)* weight function w assigns values only based on the substring selected by variable x. It is given in the input as a partial mapping μ of words to $\mathbb{Q}$ where only finitely many values are defined. The weight $w(d, \mathrm{t})$ is defined as

$$w(d,\mathrm{t}) = \begin{cases} \mu(d_{\mathrm{t}(x)}) & \text{if } d_{\mathrm{t}(x)} \text{ is in the domain of } \mu; \\ 0 & \text{otherwise.} \end{cases}$$

As we will see in Section 4, Max[VSA, Single] and Min[VSA, Single] are in FP (Corollary 4.3). Furthermore, we show that the problems Sum[$\mathcal{P}$, Single], Average[$\mathcal{P}$, Single], and q-Quantile[$\mathcal{P}$, Single] behave similarly to Count[$\mathcal{P}$], that is, they are in FP if $\mathcal{P} = \text{uVSA}$ (Corollary 4.6) and $\text{FP}^{\#\text{P}}$-complete if $\mathcal{P} = \text{VSA}$ (Theorem 4.7).

3.3.2 Polynomial-Time Weight Functions

How far can we push our tractability results? Next, we consider more general ways of mapping d-tuples into numbers. The most general class of weight functions we consider is the set of polynomial-time weight functions (Poly). A function w from Poly is given

in the input as a polynomial-time Turing Machine M that maps (d, t) pairs to values in $\mathbb{Q}$ and defines $w(d, \mathrm{t}) = M(d, \mathrm{t})$. Not surprisingly there are multiple drawbacks of having arbitrary polynomial time weight functions. The first is that all considered aggregates become intractable, even if we only consider unambiguous VSet-automata (Theorem 5.1). The second drawback is that we don't even know whether Sum and Average can be computed in $\mathrm{FP}^{\#\mathrm{P}}$ if $w \in$ Poly is a polynomial-time weight function. Therefore, we also consider two additional classes weight functions.

3.3.3 Regular Weight Functions

As the class of polynomial-time weight functions quickly leads to intractability, we focus on a restricted class that allows to examine multiple attributes, but such that we can understand the structure of the representation towards efficient algorithms. Our final classes of weight functions are based on *weighted VSet-automata* [5], which are VSet-automata that assign weights to tuples, based on a semiring. We focus on one particular semiring (the tropical semiring with min/plus) to make the presentation less abstract. One can also consider other semirings. For instance over the tropical semiring with max/plus, the complexity results are analogous to the ones we have here, with Min and Max interchanged.

To this end, let $\mathbb{Q}_\infty = \mathbb{Q} \cup \{\infty\}$. A *weighted (VSet)-automaton* is a tuple $W = (\Sigma, V, Q, I, F, \delta)$, where Σ, V, Q are as in the definition of VSet-automata, $I : Q \to \mathbb{Q}_\infty$ is the *initial weight function*; $F : Q \to \mathbb{Q}_\infty$ is the *final weight function*; and $\delta : Q \times (\Sigma \cup \Gamma_V) \times Q \to \mathbb{Q}_\infty$ is its *transition function*. Its *transitions* are the triples (p, o, q) with $\delta(p, o, q) \neq \infty$. Likewise, the *initial* (resp., *accepting*) states are those states q with $I(q) \neq \infty$ (resp., $F(q) \neq \infty$). A *run* ρ of W over a word $\mathbf{r} = \sigma_1 \cdots \sigma_n \in (\Sigma \cup \Gamma_V)^*$ is a sequence $\rho = (q_0, \sigma_1, q_1) \cdots (q_{n-1}, \sigma_n, q_n)$ of transitions. We denote the runs of W on $\mathbf{r}$ by $\mathrm{Runs}(\mathbf{r})$. The *weight* $W(\rho)$ of ρ is $I(q_0) + \delta(q_0, \sigma_1, q_1) + \cdots + \delta(q_{n-1}, \sigma_n, q_n) + F(q_n)$ and the weight $W(\mathbf{r})$ of a word $\mathbf{r}$ is $\min_{\rho \in \mathrm{Runs}(\mathbf{r})} W(\rho)$. Intuitively, a word is accepted if and only if has a finite weight. The automaton W is *unambiguous* if it satisfies conditions (1) and (2) of unambiguous VSet-automata.[1] It is *functional* if all runs are over a valid ref-word. As for ordinary VSet-automata, we also assume in this paper that weighted VSet-automata are functional.

We will consider two types of weight functions which are based on weighted VSet-automata. A *regular* (Reg) weight function w is represented by a weighted VSet-automaton W over $\Sigma \cup \Gamma_V$ and defines $w(d, \mathrm{t}) = \min_{d = \mathrm{clr}(\mathbf{r}), \mathrm{t} = \mathrm{tup}_\mathbf{r}} W(\mathbf{r})$. Given a document d and a d-tuple t, the weight $w(d, \mathrm{t})$ can be computed in polynomial time [5, Theorem 6.1].[2] The set of *unambiguous regular* (UReg) weight functions is the subset of Reg that is represented by unambiguous weighted VSet-automata.[3]

▶ **Example 3.4.** Figure 3 gives an unambiguous weighted VSet-automaton that extracts the values of three digit natural numbers from text. It can easily be extended to extract natural numbers of up to a constant number of digits by adding nondeterminism. Likewise, it is possible to extend it to extract weights as in Example 2.1. If a single variable captures a list of numbers, similar to $d_{[32,37\rangle} = 10{-}15$, one may use ambiguity to extract the minimal number represented in this range.

[1] Observe that the weight $W(\mathbf{r})$ of a word $\mathbf{r}$ is the weight of the run ρ which accepts $\mathbf{r}$.

[2] This tractability result requires functionality of W in the sense that computing $w(d, \mathrm{t})$ is NP-complete without it (Proposition 4.1). Making a weighted VSet-automaton functional is always possible and takes time polynomial in its number of states and exponential in its number of variables [5, Proposition 5.2].

[3] Testing whether a weighted VSet-automaton is in UReg can be done in PTIME: functionality can be

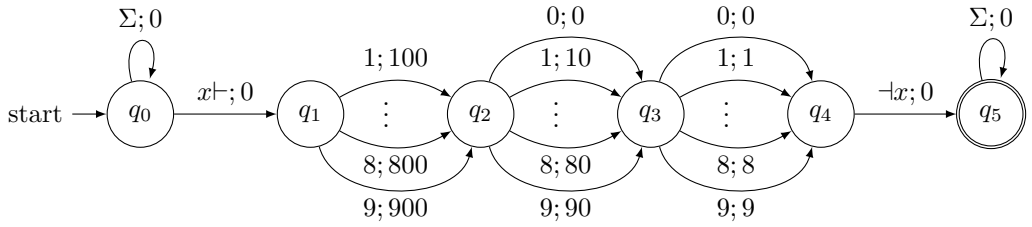


Figure 3 An unambiguous weighted VSet-automaton with initial state q_0 (with weight 0) and accepting state q_5 (with weight 0), extracting three digit natural numbers captured in variable x. Recall that the weight of a run is the sum of all its edge weights.

Our results for regular and unambiguous regular weight functions are that the situation is similar to Single when it comes to Min, Max, Sum, and Average. The main difference is that we require more unambiguity. For Max one needs unambiguity of the regular weight function and for Sum, and Average one needs unambiguity for *both* the spanner and the regular weight function to achieve tractability. For q-Quantile, the situation is different from Single in the sense that regular weight functions render the problem intractable. We refer to Table 2 for an overview.

3.4 Approximation

In the cases where exact computation of the aggregate problem is intractable, we consider the question of approximation. It turns out that there exist FPRAS's in two settings that we believe to be interesting. First, in the case of Sum and Average and single-variable weight functions, the restriction of unambiguity in the spanner can be dropped if the weight function uses only non-negative weights. Second, although q-Quantile is $\mathrm{FP}^{\#\mathrm{P}}$-complete for general VSA, it is possible to approximate the *position* of the q-quantile element in an FPRAS fashion, even with the very general polynomial-time weight functions. We discuss this problem in more detail in Section 7.

4 Single-Variable Weight Functions

We start this section by recalling that counting the number of output tuples is tractable if the spanner is functional and unambiguous (Theorem 3.2). It is well known that unambiguity is necessary in the sense that the problem becomes #P-complete without it. We next observe that functionality of the spanner is also crucial for the problem's tractability. The following proposition is heavily based on Freydenberger [9, Lemma 3.1] who showed that given a VSet-automaton A it is NP-hard to decide whether $[\![A]\!](\varepsilon) \neq \emptyset$. Based on the reduction by Freydenberger, one can also show that it the problem remains NP-hard if the VSet-automaton is unambiguous.

▶ **Proposition 4.1.** *Given a document d and non-functional VSet-automaton A, testing if $[\![A]\!](d) \neq \emptyset$ is* NP*-complete, even if A is unambiguous.*

Next, we show that Min and Max are tractable for single-variable weight functions. The reason for their tractability is that, for any fixed variable $x \in \mathrm{Vars}(A)$, the spans associated to x in output tuples can be computed in polynomial time.

tested in PTIME [5, Proposition 5.3], and testing if a functional automaton is unambiguous is also in PTIME.

▶ **Proposition 4.2.** *Let $A \in$ VSA, $x \in Vars(A)$, and $d \in \Sigma^*$. The set $\{\mathrm{t}(x) \mid \mathrm{t} \in [\![A]\!](d)\}$ can be computed in time polynomial in the sizes of A and d.*

We immediately have:

▶ **Corollary 4.3.** *MIN*[VSA, *SINGLE*] *and* *MAX*[VSA, *SINGLE*] *are in* FP.

In order to calculate aggregates like Sum, Avg, or q-Quantile, it is not sufficient to know which weights are assigned, but also the multiplicity of each weight is necessary. For general VSet-automata, this immediately results in intractability, because computing these multiplicities is hard, as we show next.

▶ **Lemma 4.4.** *Let $0 < q < 1$. Then,* *SUM*[VSA, *SINGLE*], *AVERAGE*[VSA, *SINGLE*], *and* *q-QUANTILE*[VSA, *SINGLE*] *are* $\mathrm{FP}^{\#\mathrm{P}}$*-hard, even if w is the fixed weight function*

$$w(d, \mathrm{t}) = \begin{cases} 1 & \text{if } d_{\mathrm{t}(x)} = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Proof sketch. The lower bounds are proven by reductions from the #P-complete problem COUNT[VSA]. We provide a proof sketch for AVERAGE.

Let $A \in$ VSA and $d \in \Sigma^*$. We assume w.l.o.g. that $1 \notin \Sigma$. Let $d' = d \cdot 1$. We define VSet-automaton A' as A, but change two things. First, A' only produces outputs on documents of the form $s \cdot 1$, where s is an arbitrary document. It first simulates A on s and then selects, in a fresh variable x, the symbol 1. Second, on the document d', it selects a single additional tuple t with $\mathrm{t}(y) = [1, 1\rangle$ for all its variables y. More precisely, using a regular-expression-like notation, we therefore define

$$A' = (A \cdot x\vdash \cdot 1 \cdot \dashv x) \vee (x\vdash \cdot x_1\vdash \cdots x_n\vdash \cdot \varepsilon \cdot \dashv x_n \cdots \dashv x_1 \cdot \dashv x \cdot d \cdot 1).$$

Observe that, for all $\mathrm{t} \in A'(d')$, it holds that $d_{\mathrm{t}(x)} = 1$ if and only if $\mathrm{t}{\restriction}\mathrm{Vars}(A) \in [\![A]\!](d)$. Thus, by definition of A' and w, $\mathrm{Sum}([\![A']\!], d', w) = \mathrm{Count}([\![A]\!], d)$ and $\mathrm{Count}([\![A']\!], d') = \mathrm{Count}([\![A]\!], d) + 1$. Therefore, $\mathrm{Avg}([\![A']\!], d', w) = \frac{\mathrm{Count}([\![A]\!], d)}{\mathrm{Count}([\![A]\!], d)+1}$. By solving the equation for $\mathrm{Count}([\![A]\!], d)$, it follows that $\mathrm{Count}([\![A]\!], d) = \frac{\mathrm{Avg}([\![A']\!], d', w)}{1-\mathrm{Avg}([\![A']\!], d', w)}$. This concludes the proof for AVERAGE. ◀

This result shows that for general VSet-automata, the Sum, Average, and Quantile aggregates are intractable already for very simple weight functions. However, if the spanner is *unambiguous*, we can achieve tractability. The reason is that we can compute in polynomial time the multiset $\mathbb{T}_{A,d} = \{\!\{\mathrm{t}(x) \mid \mathrm{t} \in [\![A]\!](d)\}\!\}$, where we represent the multiplicity of each span $[i, j\rangle$ (the number of tuples $\mathrm{t} \in P(d)$ such that $\mathrm{t}(x) = [i, j\rangle$) in binary.

▶ **Lemma 4.5.** *Given a VSet-automaton A and a document d, the multiset $\mathbb{T}_{A,d}$ can be computed in* FP *if $A \in$ uVSA and in* $\mathrm{FP}^{\#\mathrm{P}}$ *if $A \in$ VSA.*

It follows that all remaining aggregate functions can be efficiently computed if the spanner is given as an unambiguity functional VSet-automaton and in $\mathrm{FP}^{\#\mathrm{P}}$ otherwise.

▶ **Corollary 4.6.** *SUM*[uVSA, *SINGLE*], *AVERAGE*[uVSA, *SINGLE*], *and* *q-QUANTILE*[uVSA, *SINGLE*] *are in* FP, *for every $0 \leq q \leq 1$.*

The following theorem follows directly from Lemma 4.4 and Lemma 4.5.

▶ **Theorem 4.7.** *Let $0 < q < 1$. Then, SUM[VSA, SINGLE], AVERAGE[VSA, SINGLE], and q-QUANTILE[VSA, SINGLE$_{\mathbb{B}}$] are* FP$^{\#\mathrm{P}}$*-complete, even if w is the fixed weight function*

$$w(d, \mathrm{t}) = \begin{cases} 1 & \text{if } d_{\mathrm{t}(x)} = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we note that all tractability results in this section continue to hold for weight functions with a constant number of variables, i.e., when the weight functions are given as mappings μ of k-tuples of words to $\mathbb{Q}$, where k is a constant. We will provide proofs in the full version of the paper.

5 Polynomial-Time Weight Functions

Before we study regular weight functions, we make a few observations on the very general polynomial-time computable weight functions. For weight functions $w \in$ POLY, we assume that the number $w(d, \mathrm{t})$ is a rational number represented by its numerator and dominator, and that the function w is represented as a Turing Machine A that returns a value $A(d, \mathrm{t})$ in polynomially many steps for some fixed polynomial of choice (e.g., n^2).[4] Furthermore, to avoid any complexity due to the need to verify whether A is indeed a valid input (i.e., timely termination), we will assume that $w(d, \mathrm{t}) = 0$, if A does not produce a value within the allocated time.

We first observe that polynomial-time weight functions make all our aggregation problems intractable, which is not surprising.

▶ **Theorem 5.1.** *MIN[uVSA, POLY] and MAX[uVSA, POLY] are* OptP*-hard. Furthermore, SUM[uVSA, POLY], AVERAGE[uVSA, POLY], and q-QUANTILE[uVSA, POLY] are* FP$^{\#\mathrm{P}}$*-hard.*

In fact, all but the lower bound for MIN already hold for regular weight functions (Theorems 6.3, 6.5 and 6.6). MIN becomes tractable for regular weight functions, but it can be shown that MIN is OptP-hard for weight functions represented by weighted VSet-automata over the *numeric semiring*. Therefore, we do not require powerful weight functions for the hardness proof of MIN. Furthermore, we are able to provide OptP and FP$^{\#\mathrm{P}}$ upper bounds if the weight functions return natural numbers (or integers in the case of the FP$^{\#\mathrm{P}}$ upper bounds).

▶ **Theorem 5.2.** *MIN[VSA, POLY$_{\mathbb{N}}$] and MAX[VSA, POLY$_{\mathbb{N}}$] are in* OptP*.*

▶ **Theorem 5.3.** *SUM[VSA, POLY$_{\mathbb{Z}}$], AVERAGE[VSA, POLY$_{\mathbb{Z}}$], and q-QUANTILE[VSA, POLY$_{\mathbb{Z}}$] are in* FP$^{\#\mathrm{P}}$*, for every $0 < q < 1$.*

6 Regular Weight Functions

We now turn to REG and UREG weight functions. We first observe that SINGLE weight functions can be translated in polynomial time to equivalent UREG weight functions by a simple automata construction. So, all lower bounds for SINGLE also hold for UREG.

We show that aggregation problems for regular weight functions can often be reduced to problems about paths on weighted DAGs, where the weights come from the semiring of the weight function. To this end, a *weighted DAG* is a DAG $D = (V, E, \ell, src, snk)$, where

[4] Our complexity results are independent of the choice of this polynomial.

$\ell : E \to \mathbb{Q}_\infty$ associates a *weight* or *length* to each edge and *src* (resp., *snk*) is a unique node in V without incoming (resp., outgoing) edges. We define paths p in the obvious manner as sequences of edges and the length $\ell(p)$ of p as the sum of the lengths of its edges.

▶ **Lemma 6.1.** *Let d be a document, $A \in$ VSA and W be a weighted VSet-automaton representing $w \in$ Reg. Then we can compute, in polynomial time, a weighted DAG D such that there is a surjective mapping m from paths p from src to snk in D to tuples $\mathrm{t} \in [\![A]\!](d)$. Furthermore,*

1. *if A and W are unambiguous, then m is a bijection and*
2. *for every $\mathrm{t} \in [\![A]\!](d)$ we have $w(d, \mathrm{t}) = \min_{\{p \mid m(p) = \mathrm{t}\}} \ell(p)$.*

Proof sketch. The DAG D is obtained by a product construction between A, W, and d, such that every path from *src* to *snk* corresponds to an accepting run of W on some ref-word that represents a tuple in $[\![A]\!](d)$. This correspondence ensures that m is a bijection if A and W are unambiguous. ◀

The weighted DAG from Lemma 6.1 plays the role of a compact representation of the materialized intermediate result. It allows us to reduce Min to the shortest path problem in DAGs. If the weight function is unambiguous, Max can be reduced to the longest path problem in DAGs. Notice that, although the longest path problem is intractable in general, it is tractable for DAGs.

▶ **Theorem 6.2.** *Min*[VSA, *Reg*] *and Max*[VSA, *UReg*] *are in* FP.

The result for Max is close to the tractability frontier: if we relax the unambiguity condition in the weight function, the problem doesn't correspond to finding the longest paths in DAGs anymore and becomes intractable. In the following theorem, we restrict weight functions to natural numbers, because then we can show completeness for OptP, which is a class of functions that return natural numbers. Allowing positive and negative numbers does not fundamentally change the complexity of the problems though.

▶ **Theorem 6.3.** *Max*[uVSA, $Reg_{\mathbb{N}}$] *is* OptP*-complete.*

Since Sum and Average are already $\mathrm{FP}^{\#\mathrm{P}}$-hard for VSA spanners and Single weight functions (Theorem 4.7), they are $\mathrm{FP}^{\#\mathrm{P}}$-hard for VSA spanners and Reg/UReg weight functions as well. However, in a similar vein as in Section 4, the problems become tractable if we have unambiguity. Here, however, we require unambiguity of *both* the spanner and the representation of the weight function.

▶ **Theorem 6.4.** *Sum*[uVSA, *UReg*] *and Average*[uVSA, *UReg*] *are in* FP.

Proof sketch. Due to Lemma 6.1, these problems boil down to computing the sum of the lengths of source-to-target paths in a DAG and the average length of source-to-target paths in a DAG, respectively. Concerning Sum, we can count, for each individual edge in the DAG, the number of paths that use this edge. The sum of all output tuples is obtained by multiplying these values with the length of the edge and taking the sum over all edges. The tractability of Average then follows from the tractability of Sum and of counting the number of source-to-target paths in a DAG. ◀

Indeed, if we relax the restriction that weight functions are given as unambiguous automata, Sum and Average become $\mathrm{FP}^{\#\mathrm{P}}$-hard again.

▶ **Theorem 6.5.** *Sum*[uVSA, *Reg*] *and Average*[uVSA, *Reg*] *are* $\mathrm{FP}^{\#\mathrm{P}}$*-complete.*

The situation for q-Quantile is different from Max, Sum, and Average, since it remains hard even when both the spanner and weight function are unambiguous. The reason is that the problem reduces to counting the number of paths in a weighted DAG that are shorter than a given target weight, which is #P-complete due to Mihalak et al. [23].

▶ **Theorem 6.6.** *q-Quantile[uVSA, UReg] is $\mathrm{FP}^{\#\mathrm{P}}$-complete, for every $0 < q < 1$.*

Proof sketch. The upper bound is immediate from Theorem 6.7. For the lower bound, at the core of the quantile problem is the problem of counting up to a threshold k:

$$\textsc{Count}_{\leq k}(P, d, w) := |\{\mathrm{t} \in P(d) \mid w(d, \mathrm{t}) \leq k\}|.$$

The corresponding problem $\textsc{Count}_{\leq k}[\mathcal{P}, \mathcal{W}]$ is defined analogously to $\textsc{Sum}[\mathcal{P}, \mathcal{W}]$. It can be shown that $\textsc{Count}_{\leq k}[\mathcal{P}, \mathcal{W}]$ is #P-hard – using a reduction from #Partition, similar to Mihalak et al. [23, Theorem 1]. Using a binary search argument, $\textsc{Count}_{\leq k}$[uVSA, UReg] can be reduced to q-Quantile[uVSA, UReg], concluding the proof. ◀

Finally, we show that Sum, Average, and q-Quantile for Reg weight functions are in $\mathrm{FP}^{\#\mathrm{P}}$.

▶ **Theorem 6.7.** *Sum[VSA, Reg], Average[VSA, Reg], and q-Quantile[VSA, Reg] are in $\mathrm{FP}^{\#\mathrm{P}}$, for every $0 < q < 1$.*

7 Aggregate Approximation

Now that we have a detailed understanding on the complexity of computing exact aggregates, we want to see in which cases the result can be approximated. We only consider the situation where the exact problems are intractable and want to understand when the considered aggregation problems can be approximated by fully polynomial randomized approximation schemes (FPRAS), and when the existence of such an FPRAS would imply a collapse of the polynomial hierarchy.

▶ **Definition 7.1.** *Let f be a function that maps inputs x to rational numbers and let $\mathcal{A}$ be a probabilistic algorithm, which takes an input instance x and a parameter $\delta > 0$. Then $\mathcal{A}$ is called a* fully polynomial randomized approximation scheme *(*FPRAS*), if*

- $Pr\Big(\big|\mathcal{A}(x, \delta) - f(x)\big| \;\leq\; \delta \cdot \big|f(x)\big|\Big) \;\geq\; \frac{3}{4}$;
- *the runtime of $\mathcal{A}$ is polynomial in $|x|$ and $\frac{1}{\delta}$.*

7.1 Approximation is Hard at First Sight

For the problems Min, Max with Poly weight functions, the existence of an FPRAS would imply a collapse of the polynomial hierarchy, even when spanners are unambiguous. In the Appendix we show that the lower bound for Min already holds for weight functions represented by weighted VSet-automata over the numeric semiring.

▶ **Theorem 7.2.** *Min[uVSA, $\mathrm{Poly}_{\mathbb{N}}$] and Max[uVSA, $\mathrm{Reg}_{\mathbb{N}}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Approximation of Sum and Average is already hard for single variable weight functions. It is crucial for the hardness, however, that the weight functions can output positive and negative numbers.

▶ **Theorem 7.3.** *Sum*[VSA, *Single*$_{\{-1,1\}}$] *and Average*[VSA, *Single*$_{\{-1,1\}}$] *cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

If the spanners are unambiguous, the simplest intractable setting for Sum and Average is the one with Reg weight functions (see Table 2). Also here, the existence of an FPRAS implies a collapse of the polynomial hierarchy.

▶ **Theorem 7.4.** *Sum*[uVSA, *Reg*] *and Average*[uVSA, *Reg*] *cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

We now turn to the quantile problem. It turns out that this problem is difficult to approximate even if the weight functions only return 0 or 1.

▶ **Theorem 7.5.** *Let* $0 < q < 1$*. Then,* q*-Quantile*[VSA, *Single*$_{\mathbb{B}}$] *cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses on the second level.*

When the spanners are unambiguous, the simplest intractable case for q-Quantile is the one with UReg weight functions (see Table 2). Again, we can show that approximation is hard.

▶ **Theorem 7.6.** *Let* $0 < q < 1$*. Then,* q*-Quantile*[uVSA, *UReg*] *cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses on the second level.*

7.2 When an FPRAS is Possible

We show that Theorem 7.3 is very much on the tractability frontier: it shows that approximation is intractable if weight functions can assign 1 and -1. On the other hand, if the weight functions are restricted to *non-negative* numbers, then approximating Sum and Average is possible with an FPRAS.

▶ **Theorem 7.7.** *Sum*[VSA, *Single*$_{\mathbb{Q}_+}$] *and Average*[VSA, *Single*$_{\mathbb{Q}_+}$] *can be approximated by an* FPRAS*.*

Our second positive result is about approximating quantiles *in a positional manner*. To this end, let d be a document, P be a document spanner, w be a weight function and $0 \leq q \leq 1$ with $q \in \mathbb{Q}$. Then, for any $\delta > 0$, we say that $k \in \mathbb{Q}$ is a positional δ-approximation of q-Quantile(P, d, w) if there is a $q' \in \mathbb{Q}$, with $q - \delta \leq q' \leq q + \delta$ and $k = q'$-Quantile(P, d, w).[5]

▶ **Theorem 7.8.** *Let* $0 \leq q \leq 1$*. There is a probabilistic algorithm that calculates a positional* δ*-approximation of* q*-Quantile*[VSA, *Poly*] *with success probability at least* $\frac{3}{4}$*. Furthermore, the run time of the algorithm is polynomial in the input and* $\frac{1}{\delta}$*.*

Proof sketch. Arenas et al. [2, Corollary 4.1] showed that given a functional VSet-automaton A, one can sample tuples $\mathrm{t} \in [\![A]\!](d)$ uniformly at random with success probability at least $\frac{1}{2}$. This algorithm can be used to create a sample of $[\![A]\!](d)$ and return the q-Quantile of the sample. We show that a sample of $s \geq \frac{\ln(16)}{2\delta^2}$ tuples is sufficient to ensure that the returned quantile is indeed a positional δ-approximation of the quantile with success probability at least $\frac{7}{8}$. ◀

[5] The idea of positional quantile approximations was originally introduced by Manku et al. [19] in the context of quantile computations with limited memory.

8 Concluding Remarks

We investigated the computational complexity of common aggregate functions over regular document spanners given as regex formulas and VSet-automata. While each of the studied aggregate functions is intractable in the general case, there are polynomial-time algorithms under certain general assumptions. These include the assumption that the numerical value of the tuples is determined by a fixed variable, or that the spanner is represented as an unambiguous VSet-automaton, or the conjunction of the two assumptions. Moreover, we established quite general tractability results when randomized approximations (FPRAS) are possible. The upper bounds that we obtained for general (functional) VSet-automata immediately generalize to aggregate functions over queries that involve relational-algebra operators and string-equality conditions on top of spanners, whenever these inner queries can be *efficiently* compiled into a single VSet-automaton [10, 26]. Moreover, these upper bounds immediately generalize to allow for *grouping* (i.e., the GROUP BY operator) by computing the tuples of the grouping variables and applying the algorithms to each group separately.

We identified several interesting cases where the computation of $\alpha(P(d))$ can avoid the materialization of the exponentially large set $P(d)$, where, d is the document, P is the spanner, and α is the aggregate function. Notably, this is the case (1) for Min with general VSet-spanners and weight functions in Reg, UReg, and Single, (2) for Max with general VSet-spanners and weight functions in UReg and Single, (3) for Sum and Average with uVSA-spanners and weight functions in UReg and Single, and (4) for q-Quantile with uVSA-spanners and Single weight functions.

Yet, several basic questions are left for future investigation. A natural next step would be to seek additional useful assumptions that cast the aggregate queries tractable: Can monotonicity properties of the numerical functions lead to efficient algorithms in cases that are otherwise intractable? What are the regex formulas that can be efficiently translated into unambiguous VSet-automata (and, hence, allow to leverage the algorithms for such VSet-automata)? Another important direction is to generalize our results in a more abstract framework, such as the *Functional Aggregate Queries* (FAQ) [13], in order to provide a uniform explanation of our findings and encompass general families of aggregate functions rather than specific ones. Finally, the practical side of our work remains to be studied: How do we make our algorithms efficient in practice? How effective is the sampling approach in terms of the balancing between accuracy and execution cost? Can we accurately compute estimators of aggregate functions over (joins of) spanners within the setting of *online aggregation* [12, 16]?

References

1 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In Pablo Barceló and Marco Calautti, editors, *ICDT*, volume 127 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ICDT.2019.22.

2 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *PODS*, pages 59–73. ACM, 2019. doi:10.1145/3294052.3319704.

3 K. Y. Cockwell and I. G. Giles. Software tools for motif and pattern scanning: program descriptions including a universal sequence reading algorithm. *Computer Applications in the Biosciences*, 5(3):227–232, 1989.

4 Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-correctness in information extraction. In *PODS*, pages 149–163, 2019. doi:10.1145/3294052.3319684.

5 Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. In *ICDT*, pages 8:1–8:18, 2020.

6 A. Ehrenfeucht and H. Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.

7 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *Journal of the ACM*, 62(2):12:1–12:51, 2015. doi:10.1145/2699442.

8 Fernando Florenzano, Cristian Riveros, Martin Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, page 165–177. Association for Computing Machinery, 2018. doi:10.1145/3196959.3196987.

9 Dominik D. Freydenberger. A logic for document spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019. doi:10.1007/s00224-018-9874-1.

10 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *PODS*, pages 137–149, 2018.

11 Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In Carsten Lutz and Jean Christoph Jung, editors, *ICDT*, volume 155 of *LIPIcs*, pages 11:1–11:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICDT.2020.11.

12 Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD Conference*, pages 287–298. ACM Press, 1999.

13 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In Tova Milo and Wang-Chiew Tan, editors, *PODS*, pages 13–28. ACM, 2016. doi:10.1145/2902251.2902280.

14 Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988. doi:10.1016/0022-0000(88)90039-6.

15 Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. SystemT: A system for declarative information extraction. *SIGMOD Record*, 37(4):7–13, 2008. doi:10.1145/1519103.1519105.

16 Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *SIGMOD Conference*, pages 615–629. ACM, 2016.

17 Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. SVM based learning system for information extraction. In *Deterministic and Statistical Methods in Machine Learning*, volume 3635 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2004.

18 Yunyao Li, Frederick Reiss, and Laura Chiticariu. SystemT: A declarative information extraction system. In *ACL*, pages 109–114. ACL, 2011.

19 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD Conference*, page 426–435. Association for Computing Machinery, 1998. doi:10.1145/276304.276342.

20 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *PODS*, pages 125–136, 2018.

21 Franz Mayr and Sergio Yovine. Regular inference on artificial neural networks. In *CD-MAKE*, volume 11015 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2018.

22 Joshua J. Michalenko, Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. Representing formal languages: A comparison between finite automata and recurrent neural networks. In *ICLR (Poster)*. OpenReview.net, 2019.

23 Matús Mihalák, Rastislav Srámek, and Peter Widmayer. Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory Comput. Syst.*, 58(1):45–59, 2016. doi:10.1007/s00224-014-9571-7.

24 A. Neuwald and P. Green. Detecting patterns in protein sequences. *Journal of Molecular Biology*, 239:698–712, 1994.

25 Galia Nordon, Gideon Koren, Varda Shalev, Benny Kimelfeld, Uri Shalit, and Kira Radinsky. Building causal graphs from medical literature and electronic medical records. In *AAAI*, pages 1102–1109. AAAI Press, 2019.

26 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *PODS*, pages 320–334. ACM, 2019. doi:10.1145/3294052.3319699.

27 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *ICDT*, volume 127 of *LIPIcs*, pages 13:1–13:18, 2019.

28 Hoifung Poon and Pedro M. Domingos. Joint inference in information extraction. In *AAAI*, pages 913–918. AAAI Press, 2007.

29 Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, 2017.

30 Robert P. Schumaker and Hsinchun Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Trans. Inf. Syst.*, 27(2):12:1–12:19, 2009. doi:10.1145/1462198.1462204.

31 Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *PVLDB*, 8(11):1310–1321, 2015. doi:10.14778/2809974.2809991.

32 Charles A. Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

33 Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*. MIT Press, Cambridge, MA, USA, 1991.

34 Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *ICML*, volume 80, pages 5244–5253. JMLR.org, 2018.

A Algebraic Operations on Spanners

We will now recall some definitions of algebraic operations on spanners. To this end, we start with some basic definitions. Two d-tuples t_1 and t_2 are *compatible* if they agree on every common variable, i.e., $\mathrm{t}_1(x) = \mathrm{t}_2(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_1) \cap \mathrm{Vars}(\mathrm{t}_2)$. In this case, define $\mathrm{t}_1 \bowtie \mathrm{t}_2$ as the tuple with $\mathrm{Vars}(\mathrm{t}_1 \bowtie \mathrm{t}_2) = \mathrm{Vars}(\mathrm{t}_1) \cup \mathrm{Vars}(\mathrm{t}_2)$ such that $(\mathrm{t}_1 \bowtie \mathrm{t}_2)(x) = \mathrm{t}_1(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_1)$ and $(\mathrm{t}_1 \bowtie \mathrm{t}_2)(x) = \mathrm{t}_2(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_2)$. Note that the following operators are the same as those defined by Fagin et al [7].

▶ **Definition A.1** (Algebraic Operations on Spanners). *Let P, P_1, P_2 be spanners and let $d \in \Sigma^*$ be a document.*

- Union. *The union $P = P_1 \cup P_2$ is defined when $Vars(P_1) = Vars(P_2)$. In that case, $P(d) = P_1(d) \cup P_2(d)$.*
- Projection. *The projection $P = \pi_Y P_1$ is defined by $P(d) = \{\mathrm{t}{\restriction}Y \mid \mathrm{t} \in P_1(d)\}$. Recall that $\mathrm{t}{\restriction}Y$ denotes the restriction of t to the variables in Y.*
- Natural Join. *The (natural) join $P = P_1 \bowtie P_2$ is defined such that $P(d)$ consists of all tuples $\mathrm{t}_1 \bowtie \mathrm{t}_2$ such that $\mathrm{t}_1 \in P_1(d)$, $\mathrm{t}_2 \in P_2(d)$, and t_1 and t_2 are compatible.*

B Regex-formulas

A *regex-formula (over Σ)* is a regular expression that may include variables (called capture variables). Formally, we define the syntax with the recursive rule

$$\alpha := \emptyset \mid \varepsilon \mid \sigma \mid (\alpha \vee \alpha) \mid (\alpha \cdot \alpha) \mid \alpha^* \mid x\vdash \alpha \dashv x\,,$$

where $\sigma \in \Sigma$ and $x \in \mathrm{Vars}$. We use α^+ as a shorthand for $\alpha \cdot \alpha^*$ and Σ as a shorthand for $\bigvee_{\sigma\in\Sigma} \sigma$. The set of variables that occur in α is denoted by $\mathrm{Vars}(\alpha)$ and the size $|\alpha|$ is defined as the number of symbols in α.

Every regex-formula can be interpreted as a generator of a (regular) ref-word language $\mathcal{R}(\alpha)$ over the extended alphabet $\Sigma \cup \Gamma_{\mathrm{Vars}(\alpha)}$. If α is of the form $x\vdash \beta \dashv x$, then $\mathcal{R}(\alpha) := \{x\vdash\} \cdot \mathcal{R}(\beta) \cdot \{\dashv x\}$. Otherwise, $\mathcal{R}(\alpha)$ is defined as the language $\mathcal{L}(\alpha)$, that is $\mathcal{R}(\emptyset) := \emptyset$, $\mathcal{R}(a) := \{a\}$ for every $a \in \Sigma \cup \{\varepsilon\}$, $\mathcal{R}(\alpha \vee \beta) := \mathcal{R}(\alpha) \cup \mathcal{R}(\beta)$, $\mathcal{R}(\alpha \cdot \beta) := \mathcal{R}(\alpha) \cdot \mathcal{R}(\beta)$, $\mathcal{R}(\alpha^*) := \{\mathcal{R}(\alpha)^i \mid i \geq 0\}$.

Notice that $\mathcal{R}(\alpha)$ can contain ref-words in which the same variable is used multiple times. By $\mathcal{VR}(\alpha)$ we denote the set of ref-words in $\mathcal{R}(\alpha)$ that are valid and by $\mathcal{VR}_{\mathrm{Vars}(\alpha)}(\alpha)$ we denote the set of ref-words in $\mathcal{R}(\alpha)$ that are valid for $\mathrm{Vars}(\alpha)$. For example, if $\alpha = (x\vdash a \dashv x)^*$, then $\mathcal{VR}(\alpha) = \{\varepsilon, x\vdash a\dashv x\}$ and $\mathcal{VR}_{\mathrm{Vars}(\alpha)}(\alpha) = \{x\vdash a\dashv x\}$. For every document $d \in \Sigma^*$, we define $\mathcal{VR}(\alpha, d) := \mathcal{VR}(\alpha) \cap \mathcal{VR}(d)$. In other words, $\mathcal{VR}(\alpha, d)$ contains exactly those valid ref-words from $\mathcal{VR}(\alpha)$ that clr maps to d. Finally, the spanner $[\![\alpha]\!]$ is the one that maps every document $d \in \Sigma^*$ to the following set of tuples:

$$[\![\alpha]\!](d) := \{\mathrm{tup}_{\mathbf{r}} \mid \mathbf{r} \in \mathcal{VR}(\alpha, d)\}$$

We will sometimes denote the set of tuples $[\![\alpha]\!](d)$ by $\alpha(d)$ to simplify notation. We say that a regex-formula is *functional* if $\mathcal{R}(\alpha) = \mathcal{VR}_{\mathrm{Vars}(\alpha)}(\alpha)$, that is, every ref-word in $\mathcal{R}(\alpha)$ is valid for $\mathrm{Vars}(\alpha)$. As for VSet-automata, in this paper, we assume that regex-formulas are functional. The set of all functional regex-formulas is denoted by RGX.

C Some Background on Complexity

We will recall the definitions for some of the complexity classes we will use in the following sections, closely following [33]. The class FP is the set of all functions that are computable in polynomial time. A *counting Turing Machine* is an non-deterministic Turing Machine whose output for a given input is the number of accepting computations for that input. The class #P is the set of all functions that are computable by polynomial-time counting Turing Machines. A problem X is #P-*hard* if there are polynomial time Turing reductions to it from all problems in #P. If in addition $X \in$ #P, we say that X is #P-complete. The class $\mathrm{FP}^{\#\mathrm{P}}$ is the set of all functions that are computable in polynomial time by an oracle Turing Machine with an #P oracle. It is easy to see that, under Turing reductions, a problem is hard for the class #P if and only if it is hard for $\mathrm{FP}^{\#\mathrm{P}}$. For counting problems, use Cook reductions, also known as Turing reductions. Under these reductions, counting the number of satisfying assignments of a DNF formula or the number of words accepted by some NFA is #P-complete.

The class OptP is the set of all functions computable by taking the maximum output values over all accepting computations of a polynomial-time non-deterministic Turing Machine that outputs natural numbers. Assume that Γ is the Turing Machine alphabet. Let $f, g : \Gamma^* \to \mathbb{N}$

be functions. A *metric reduction*, as introduced by Krentel [14], from f to g is a pair of polynomial-time computable functions T_1, T_2, where $T_1 : \Gamma^* \to \Gamma^*$ and $T_2 : \Gamma^* \times \mathbb{N} \to \mathbb{N}$, such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Gamma^*$.

The class BPP is the set of all decision problems solvable in polynomial time by a probabilistic Turing Machine in which the answer always has probability at least $\frac{1}{2} + \delta$ of being correct for some fixed $\delta > 0$.

Answer Counting Under Guarded TGDs

Cristina Feier
Department of Computer Science, Universität Bremen, Germany

Carsten Lutz
Department of Computer Science, Universität Bremen, Germany

Marcin Przybyłko
Department of Computer Science, Universität Bremen, Germany

Abstract

We study the complexity of answer counting for ontology-mediated queries and for querying under constraints, considering conjunctive queries and unions thereof (UCQs) as the query language and guarded TGDs as the ontology and constraint language, respectively. Our main result is a classification according to whether answer counting is fixed-parameter tractable (FPT), W[1]-equivalent, #W[1]-equivalent, #W[2]-hard, or #A[2]-equivalent, lifting a recent classification for UCQs without ontologies and constraints due to Dell et al. [19]. The classification pertains to various structural measures, namely treewidth, contract treewidth, starsize, and linked matching number. Our results rest on the assumption that the arity of relation symbols is bounded by a constant and, in the case of ontology-mediated querying, that all symbols from the ontology and query can occur in the data (so-called full data schema). We also study the meta-problems for the mentioned structural measures, that is, to decide whether a given ontology-mediated query or constraint-query specification is equivalent to one for which the structural measure is bounded.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Ontology-Mediated Querying, Querying under Constraints, Answer Counting, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.11

Related Version *Full Version*: `https://arxiv.org/abs/2101.03058` [23]

Funding This research was funded by ERC consolidator grant 647289 CODA and by DFG project QTEC.

Acknowledgements We thank the anonymous reviewers for useful comments.

1 Introduction

Tuple-generating dependencies (TGDs) are a prominent formalism for formulating database constraints. A TGD states that if certain facts are true, then certain other facts must be true as well. This can be interpreted in different ways. In *ontology-mediated querying*, TGDs give rise to ontology languages and are used to derive new facts in addition to those that are present in the database. This makes it possible to obtain additional answers if the data is incomplete and also enriches the vocabulary that is available for querying. In a more classical setup that we refer to as *querying under constraints*, TGDs are used as integrity constraints on the database, that is, a TGD expresses the promise that if certain facts are present in the database, then certain other facts are present as well. Integrity constraints are relevant to query optimization as they might enable the reformulation of a query into a "simpler" one. TGDs generalize a wide range of other integrity constraints, which was the original reason for introducing them [1].

When unrestricted TGDs are used as an ontology language, ontology-mediated querying is undecidable even for unary queries that consist of a single atom [12]. This has led to intense research on identifying restricted forms of TGDs that regain decidability, see [3, 12, 13, 30] and references therein. In this paper, we consider guardedness as a basic and robust such restriction: a TGD is guarded if some body atom, the guard, contains all body variables [12]. Guarded TGDs are useful also for formalizing integrity constraints. For example, the important class of referential integrity constraints (also known as inclusion dependencies) is a special case of guarded TGDs.

While being decidable, both ontology-mediated querying and querying under constraints with guarded TGDs is computationally intractable. Let us make this precise for query evaluation, i.e. the problem to decide, given a database, a query, and a candidate answer, whether the candidate is indeed an answer. We use $(\mathbb{G}, \mathbb{CQ})$ to denote the language of ontology-mediated queries $(\mathcal{O}, \mathbf{S}, q)$ that consist of an ontology $\mathcal{O}$ which is a set of guarded TGDs, a data schema $\mathbf{S}$, and a conjunctive query (CQ) q. As usual, $\mathbf{S}$ contains the relation names that can be used in the data while both the ontology and query can also use additional names. Evaluating ontology-mediated queries (OMQs) from $(\mathbb{G}, \mathbb{CQ})$ is 2ExpTime-complete in combined complexity. The same holds for $(\mathbb{G}, \mathbb{UCQ})$ where the queries are unions of CQs (UCQs) [12]. For querying under constraints, we consider constraint query specifications (CQSs) of the form $(\mathcal{T}, \mathbf{S}, q)$ where $\mathcal{T}$ is a set of integrity constraints and q is a query, both over schema $\mathbf{S}$. Overloading notation, we use $(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$ also to denote the class of CQSs in which the constraints are guarded TGDs and the queries are (U)CQs; it will always be clear from the context whether $(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$ denotes an OMQ language or a class of CQSs. Query evaluation for CQSs from $(\mathbb{G}, \mathbb{CQ})$ and $(\mathbb{G}, \mathbb{UCQ})$ is NP-complete.

In this paper, we are interested in counting the number of answers to OMQs and to queries posed under integrity constraints, with an emphasis on the limits of efficiency from the viewpoint of parameterized complexity theory. Counting the number of answers is important to inform the user when there are too many answers to compute all of them, and it is supported by almost every data management system. It is also a fundamental operation in data analytics and in decision support where often the count is more important than the actual answers. Despite its relevance, however, the problem has received little attention in ontology-mediated querying and querying under constraints, see [29, 28, 9, 14] for some notable exceptions.

We equate efficiency with fixed-parameter tractability (FPT), the parameter being the size of the OMQ and of the CQS, respectively. Evaluating Boolean queries is W[1]-hard both for ontology-mediated querying in $(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$ and for querying under constraints in $(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$ [5], and therefore answer counting (which is the same problem as query evaluation for Boolean queries) is in general not fixed-parameter tractable unless FPT = W[1]. The main question that we ask is: how can we characterize the parameterized complexity of answer counting for classes of OMQs or CQSs $\mathbb{C} \subseteq (\mathbb{G}, (\mathbb{U})\mathbb{CQ})$ and, most importantly, for which such classes $\mathbb{C}$ can we count answers in FPT? The classes $\mathbb{C}$ will primarily be defined in terms of structural restrictions of the (U)CQ, but will also take into account the interplay between the ontology/constraints and the (U)CQ. Note that PTime combined complexity, a (significant) strengthening of FPT, cannot be obtained by structural restrictions on the UCQ in ontology-mediated querying with $(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$ because evaluating Boolean OMQs is 2ExpTime-complete already for unary single atom queries. For querying under constraints, in contrast, PTime combined complexity is not out of reach and in the case of query evaluation can in fact sometimes be attained in $(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$ [8, 7].

A seminal result due to Grohe states that a recursively enumerable class $\mathbb{C}$ of CQs can be evaluated in FPT if and only if there is a constant that bounds the treewidths of CQs in $\mathbb{C}$, modulo equivalence [26]. Here, treewidth of a CQ q means the treewidth of the Gaifman graph of q after dropping all answer variables. The result rests on the assumptions that FPT $\neq$ W[1] and that the arity of relation symbols is bounded by a constant, which we shall also assume throughout this article. Grohe's result extends to UCQs in the expected way, that is, the characterization for UCQs is in terms of the maximum treewidth of the constituting CQs modulo equivalence, assuming w.l.o.g. that there are no containment relations among them. An adaptation of Grohe's proof was used by Dalmau and Jonsson to show that a class $\mathbb{C}$ of CQs without quantified variables admits answer counting in FPT if and only if the treewidths of CQs in $\mathbb{C}$ is bounded by a constant [18]. In a series of papers by Pichler and Skritek [32], Durand and Mengel [20, 21], Chen and Mengel [16, 17], and Dell et al. [19], this was extended to a rather detailed classification of the parameterized complexity of answer counting for classes of CQs and UCQs that may contain both answer variables and quantified variables. The characterization uses treewidth, which now refers to the entire Gaifman graph including the answer variables. It also refers to the additional structural measures of contract treewidth, starsize,[1] and linked matching number. It links boundedness of these measures by a constant, modulo equivalence, to the relevant complexities, which turn out to be FPT, W[1]-equivalence, #W[1]-equivalence, #W[2]-hardness, and #A[2]-equivalence. Here, we speak of "equivalence" rather than of "completeness" to emphasize that hardness is defined in terms of (parameterized counting) Turing (fpt-)reductions.

The main results of this article are classifications of the complexity of answer counting for classes of ontology-mediated queries from $(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$, assuming that the data schema contains all symbols used in the ontology and query, and for classes of constraint query specifications from $(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$. Our classifications parallel the one for the case without TGDs, involve the same five complexities mentioned above, and link them to the same structural measures. However, there is a twist. The ontology interacts with all of the mentioned structural measures in the sense that for each measure, there is a class of CQs $\mathbb{C}$ and an ontology $\mathcal{O}$ such that the measure is unbounded for $\mathbb{C}$ modulo equivalence while there is a constant k such that each OMQ $(\mathcal{O}, \mathbf{S}, q)$, $q \in \mathbb{C}$, is equivalent to an OMQ $(\mathcal{O}, \mathbf{S}, q')$ with the measure of q' bounded by k. A similar effect can be observed for querying under constraints. We can thus not expect to link the complexity of a class $\mathbb{C}$ of OMQs to the structural measures of the actual queries in the OMQs. Instead, we consider a certain class of CQs that we obtain from the OMQs in $\mathbb{C}$ by first rewriting away the existential quantifiers in TGD heads in the ontology, then taking the CQs that occur in the resulting OMQs, combining them conjunctively guided by the inclusion-exclusion principle, next chasing them with the ontology (which is a finite operation due to the first step), and then taking the homomorphism core. The structural measures of the resulting class of CQs turn out to determine the complexity of answer counting for the original class of OMQs $\mathbb{C}$. Interestingly, the same is also true for classes of constraint query specifications and thus the characterizations for OMQs and for CQSs coincide. We in fact establish the latter by mutual reduction between answer counting for OMQs and answer counting for CQSs.

Inspired by our complexity classifications, we also study the meta problems to decide whether a given query is equivalent to a query in which some selected structural measures are small, and to construct the latter query if it exists. We do this both for ontology-mediated

[1] The measure is called dominating starsize in [19] and strict starsize in [16]. We only speak of starsize. Note that this is not identical to the original notion of starsize from [20, 21].

queries and for queries under constraints, considering all four measures that are featured in the classifications (and sets thereof). We start with querying under constraints where we are able to obtain decidability results in all relevant cases. These results can also be applied to ontology-mediated querying when (i) the data schema contains all symbols used in the ontology and query and (ii) we require that the ontology used in the OMQ cannot be replaced with a different one. For contract treewidth and starsize, we additionally show that it is never necessary to modify the ontology to attain equivalent OMQs with small measures, and we provide decidability results without assumptions (i) and (ii). We also observe that treewidth behaves differently in that modifying the ontology might result in smaller measures. Deciding the meta problems for the measure of treewidth is left open as an interesting and non-trivial open problem.

Some proofs are deferred to the appendix of the long version of this paper [23], available at `https://arxiv.org/abs/2101.03058`.

Related Work. The complexity of ontology-mediated querying has been a subject of intense study from various angles, see for example [10, 11, 33] and references therein. The parameterized complexity of evaluating ontology-mediated queries has been studied in [6, 5]. In [5], it is shown that query evaluation in FPT coincides with bounded treewidth modulo equivalence in $(\mathbb{G}, \mathbb{UCQ})$ when the arity of relation symbols is bounded by a constant. An FPT upper bound for querying under constraints that are guarded TGDs has been established in [8, 7] for CQs that have bounded generalized hypertreewidth modulo equivalence. That paper also studies the meta problems for querying under constraints that are guarded TGDs and for the measure of generalized hypertree width. A related topic is query containment under constraints, see for example [15, 27, 24].

2 Preliminaries

For an integer $n \geq 1$, we use $[n]$ to denote the set $\{1, \ldots, n\}$. To indicate the cardinality of a set S, we may write $\#S$ or $|S|$.

Relational Databases. A *schema* $\mathbf{S}$ is a set of relation symbols R with associated arity $\mathsf{ar}(R) \geq 0$. We write $\mathsf{ar}(\mathbf{S})$ for $\max_{R \in \mathbf{S}}\{\mathsf{ar}(R)\}$. An $\mathbf{S}$-*fact* is an expression of the form $R(\bar{c})$, where $R \in \mathbf{S}$ and $\bar{c}$ is an $\mathsf{ar}(R)$-tuple of constants. An $\mathbf{S}$-*instance* is a (possibly infinite) set of $\mathbf{S}$-facts and an $\mathbf{S}$-*database* is a finite $\mathbf{S}$-instance. We write $\mathsf{adom}(I)$ for the set of constants in an instance I. For a set $S \subseteq \mathsf{adom}(I)$, we denote by $I_{|S}$ the restriction of I to facts that mention only constants from S. A *homomorphism* from I to an instance J is a function $h : \mathsf{adom}(I) \to \mathsf{adom}(J)$ such that $R(h(\bar{c})) \in J$ for every $R(\bar{c}) \in I$. A database D' is obtained from a database D by *cloning constants* if $D' \supseteq D$ can be constructed by choosing $a_1, \ldots, a_n \in \mathsf{adom}(D)$ and positive integers $m_1, \ldots, m_n$, reserving fresh constants $a_1^{i_1}, \ldots, a_n^{i_n}$, $1 \leq i_\ell \leq m_\ell$ for $1 \leq \ell \leq n$, and adding to D each atom $R(\bar{a}')$ that can be obtained from some $R(\bar{a}) \in D$ by replacing each occurrence of a_i, $1 \leq i \leq n$, with a_i^j for some j with $1 \leq j \leq m_i$.

CQs and UCQs. A *conjunctive query* (CQ) $q(\bar{x})$ over a schema $\mathbf{S}$ is a first-order formula of the form $\exists \bar{y}\, \varphi(\bar{x}, \bar{y})$ where φ is a conjunction of *relational atoms* $R_i(\bar{x}_i)$ with $R_i \in \mathbf{S}$ and $\bar{x}_i$ a tuple of variables of length $\mathsf{ar}(R_i)$ and *equality atoms* $x_1 = x_2$. We require that only variables from $\bar{x}$ appear in equality atoms. With $\mathsf{var}(q)$, we denote the set of variables that occur in $\bar{x}$ or in $\bar{y}$. Whenever convenient, we identify a conjunction of atoms with a set of atoms. When we are not interested in order and multiplicity, we treat $\bar{x}$ as a set of variables. We write $\mathbb{CQ}$ for the class of CQs.

Every CQ q can be naturally seen as a database D_q, known as the *canonical database* of q, obtained by dropping the existential quantifier prefix and the equality atoms, and viewing variables as constants. A *homomorphism* h from a CQ q to an instance I is a homomorphism from D_q to I such that $x = y \in q$ implies $h(x) = h(y)$. A tuple $\bar{c} \in \mathsf{adom}(I)^{|\bar{x}|}$ is an *answer* to Q on I if there is a homomorphism h from q to I with $h(\bar{x}) = \bar{c}$. The *evaluation of* $q(\bar{x})$ *on* I, denoted $q(I)$, is the set of all answers to Q on I.

A *union of conjunctive queries* (UCQ) over a schema $\mathbf{S}$ is a first-order formula of the form $q(\bar{x}) := q_1(\bar{x}) \vee \cdots \vee q_n(\bar{x})$, where $n \geq 1$, and $q_1(\bar{x}), \ldots, q_n(\bar{x})$ are CQ over $\mathbf{S}$. We refer to the variables in $\bar{x}$ as the *answer variables* of q and the *arity* of q is defined as the number of its answer variables. The *evaluation* of q on an instance I, denoted $q(I)$, is the set of tuples $\bigcup_{i\in[n]} q_i(I)$. We write $\mathbb{UCQ}$ for the class of UCQs. A (U)CQ of arity zero is called *Boolean*. The only possible answer to a Boolean query is the empty tuple. For a Boolean (U)CQ q, we may write $I \models q$ if $q(I) = \{()\}$ and $I \not\models q$ otherwise.

Let $q_1(\bar{x})$ and $q_2(\bar{x})$ be two UCQs over the same schema $\mathbf{S}$. We say that q_1 is *contained* in q_2, written $q_1 \subseteq_{\mathbf{S}} q_2$, if $q_1(D) \subseteq q_2(D)$ for every $\mathbf{S}$-database D. Moreover, q_1 and q_2 are *equivalent*, written $q_1 \equiv_{\mathbf{S}} q_2$, if $q_1 \subseteq_{\mathbf{S}} q_2$ and $q_2 \subseteq_{\mathbf{S}} q_1$.

A CQ $q(\bar{x})$ is a *core* if every homomorphism h from q to D_q with $h(\bar{x}) = \bar{x}$ is surjective. Every CQ $q(\bar{x})$ is equivalent to a CQ $p(\bar{x})$ that is a core and can be obtained from q by dropping atoms. In fact, p is unique up to isomorphism and we call it the *core of* q. For a UCQ q, we use $\mathsf{core}(q)$ to denote the disjunction whose disjuncts are the cores of the CQs in q.

For a UCQ q, but also for any other syntactic object q, we use $||q||$ to denote the number of symbols needed to write q as a word over a suitable alphabet.

Our main interest is in the complexity of counting the number of answers. Every choice of a query language $\mathbb{Q}$, such as $\mathbb{CQ}$ and $\mathbb{UCQ}$, and a class of databases $\mathbb{D}$ gives rise to the following answer counting problem:

PROBLEM :	$\mathsf{AnswerCount}(\mathbb{Q}, \mathbb{D})$
INPUT :	A query $q \in \mathbb{Q}$ over some schema $\mathbf{S}$ and an $\mathbf{S}$-database $D \in \mathbb{D}$
OUTPUT :	$\#q(D)$

Our main interest is in the parameterized version of the above problem where we generally assume that the parameter is the size of the input query, see below for more details. When $\mathbb{D}$ is the class of all databases, we simply write $\mathsf{AnswerCount}(\mathbb{Q})$.

Treewidth. Treewidth is a widely used notion that measures the degree of tree-likeness of a graph. Let $G = (V, E)$ be an undirected graph. A *tree decomposition* of G is a pair $\delta = (T_\delta, \chi)$, where $T_\delta = (V_\delta, E_\delta)$ is a tree, and χ is a labeling function $V_\delta \to 2^V$, i.e., χ assigns a subset of V to each node of T_δ, such that:

1. $\bigcup_{t\in V_\delta} \chi(t) = V$,
2. if $\{u, v\} \in E$, then $u, v \in \chi(t)$ for some $t \in V_\delta$,
3. for each $v \in V$, the set of nodes $\{t \in V_\delta \mid v \in \chi(t)\}$ induces a connected subtree of T_δ.

The *width* of δ is the number $\max_{t\in V_\delta}\{|\chi(t)|\} - 1$. If the edge set E of G is non-empty, then the *treewidth* of G is the minimum width over all its tree decompositions; otherwise, it is defined to be one. Each instance I is associated with an undirected graph (without self loops) $G_I = (V, E)$, called the *Gaifman graph* of I, defined as follows: $V = \mathsf{adom}(I)$, and $\{a, b\} \in E$ iff there is a fact $R(\bar{c}) \in I$ that mentions both a and b. The *treewidth* of I is the treewidth of G_I.

TGDs, Guardedness. A *tuple-generating dependency* (TGD) T over $\mathbf{S}$ is a first-order sentence of the form $\forall\bar{x}\forall\bar{y}\left(\phi(\bar{x},\bar{y}) \to \exists\bar{z}\,\psi(\bar{x},\bar{z})\right)$ such that $\exists\bar{y}\,\phi(\bar{x},\bar{y})$ and $\exists\bar{z}\,\psi(\bar{x},\bar{z})$ are CQs without equality atoms. As a special case, we also allow $\phi(\bar{x},\bar{y})$ to be the empty conjunction, i.e. logical truth, denoted by true. For simplicity, we write T as $\phi(\bar{x},\bar{y}) \to \exists\bar{z}\,\psi(\bar{x},\bar{z})$. We call ϕ and ψ the *body* and *head* of T, denoted $\mathsf{body}(T)$ and $\mathsf{head}(T)$, respectively. An instance I over $\mathbf{S}$ *satisfies* T, denoted $I \models T$, if $q_\phi(I) \subseteq q_\psi(I)$. It *satisfies* a set of TGDs S, denoted $I \models S$, if $I \models T$ for each $T \in S$. We then also say that I is a *model* of S. We write $\mathbb{TGD}$ to denote the class of all TGDs.

A TGD T is *guarded* if $\mathsf{body}(T)$ is true or there exists an atom α in its body that contains all variables that occur in $\mathsf{body}(T)$ [12]. Such an atom α is the *guard* of T, denoted $\mathsf{guard}(T)$. We write $\mathbb{G}$ for the class of guarded TGDs. A TGD T is *full* if the tuple $\bar{z}$ of variables is empty. We use $\mathbb{FULL}$ to denote the class of full TGDs and shall often refer to $\mathbb{G} \cap \mathbb{FULL}$, the class of TGDs that are both guarded and full. Note that this class is essentially the class of Datalog programs with guarded rule bodies.

We next introduce the well-known chase procedure for making explicit the consequences of a set of TGDs [31, 27, 22, 12]. We first define a single chase step. Let I be an instance over a schema $\mathbf{S}$ and $T = \phi(\bar{x},\bar{y}) \to \exists\bar{z}\,\psi(\bar{x},\bar{z})$ a TGD over $\mathbf{S}$. We say that T is *applicable* to a tuple $(\bar{c},\bar{c}')$ of constants in I if $\phi(\bar{c},\bar{c}') \subseteq I$. In this case, *the result of applying T in I at* $(\bar{c},\bar{c}')$ is the instance $J = I \cup \psi(\bar{c},\bar{c}'')$, where $\bar{c}''$ is the tuple obtained from $\bar{z}$ by simultaneously replacing each variable z with a fresh distinct constant that does not occur in I. We describe such a single chase step by writing $I \xrightarrow{T,(\bar{c},\bar{c}')} J$. Let I be an instance and S a finite set of TGDs. A *chase sequence for I with S* is a sequence of chase steps

$$I_0 \xrightarrow{T_0,(\bar{c}_0,\bar{c}'_0)} I_1 \xrightarrow{T_1,(\bar{c}_1,\bar{c}'_1)} I_2 \dots$$

such that (1) $I_0 = I$, (2) $T_i \in S$ for each $i \geq 0$, and (3) $J \models S$ with $J = \bigcup_{i\geq 0} I_i$. The instance J is the (potentially infinite) *result* of this chase sequence, which always exists. The chase sequence is *fair* if whenever a TGD $T \in S$ is applicable to a tuple $(\bar{c},\bar{c}')$ in some I_i, then $I_j \xrightarrow{T,(\bar{c},\bar{c}')} I_{j+1}$ is part of the sequence for some $j \geq i$. Note that our chase is oblivious, that is, a TGD is triggered whenever its body is satisfied, even if also its head is already satisfied. As a consequence, every fair chase sequence for I with S leads to the same result, up to isomorphism. Thus, we can refer to *the* result of chasing I with S, denoted $\mathsf{ch}_S(I)$.

▶ **Lemma 1.** *Let S be a finite set of TGDs and I an instance. Then for every model J of S with $I \subseteq J$, there is a homomorphism h from $\mathsf{ch}_S(I)$ to J that is the identity on $\mathsf{adom}(I)$.*

For sets S of TGDs from $\mathbb{G} \cap \mathbb{FULL}$, we may also chase a CQ $q(\bar{x})$ with S, denoting the result with $\mathsf{ch}_S(q)$. What we mean is the (finite!) result of chasing database D_q with S and viewing the resulting as a CQ with answer variables $\bar{x}$.

Parameterized Complexity. A *counting problem* over a finite alphabet Λ is a function $P : \Lambda^* \to \mathbb{N}$ and a *parameterized counting problem* over Λ is a pair (P,κ), with P a counting problem over Λ and κ the *parameterization* of P, a function $\kappa : \Lambda^* \to \mathbb{N}$ that is computable in PTime. An example of a parameterized counting problem is #pClique in which P maps (a suitable encoding of) each pair (G,k) with G an undirected graph and $k \geq 0$ a clique size to the number of k-cliques in G, and where $\kappa(G,k) = k$. Another example is #pDomSet where P maps each pair (G,k) to the number of dominating sets of size k, and where again $\kappa(G,k) = k$.

A counting problem P is a *decision problem* if the range of P is $\{0, 1\}$, and a *parameterized decision problem* is defined accordingly. An example of a parameterized decision problem is pClique in which P maps each pair (G, k) to 1 if the undirected graph G contains a k-clique and to 0 otherwise, and where $\kappa(G, k) = k$.

A parameterized problem (P, κ) is *fixed-parameter tractable* (fpt) if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $P(x)$ can be computed in time $|x|^{O(1)} \cdot f(\kappa(x))$ for all inputs x. We use FPT to denote the class of all parameterized counting problems that are fixed-parameter tractable.

A *Turing fpt-reduction* from a parameterized counting problem (P_1, κ_1) to a parameterized counting problem (P_2, κ_2) is an algorithm that computes P_1 with oracle access to P_2, runs within the time bounds of fixed parameter tractability for (P_1, κ_1), and when started on input x only makes oracle calls with argument y such that $\kappa_2(y) \leq f(\kappa_1(x))$, for some computable function f. The reduction is called a *parsimonious fpt-reduction* if only a single oracle call is made at the end of the computation and its output is then returned as the output of the algorithm without any further modification.

A parameterized counting problem (P, κ) is #W[1]-*easy* if it can be reduced to #pClique and it is #W[1]-*hard* if #pClique reduces to (P, κ), both in terms of Turing fpt-reductions. W[1]-easiness and -hardness are defined analogously, but using pClique in place of #pClique, and likewise for #W[2] and #pDomSet, and for #A[2] and the parameterized problem of counting the answers to CQs, the parameter being the size of the CQ. For $C \in \{\text{W}[1], \#\text{W}[1], \#\text{W}[2], \#\text{A}[2]\}$, (P, κ) is C-*equivalent* if it is C-easy and C-hard. Note that we follow [16, 19] in defining both easiness and hardness in terms of Turing fpt-reductions; stronger notions would rely on parsimonious fpt-reductions [25].

3 The Classification Without TGDs

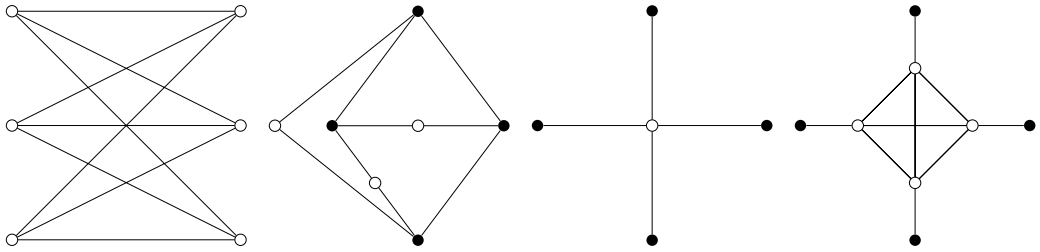
In the series of papers [20, 21, 16, 17, 19], the parameterized complexity of answer counting is studied for classes of CQs and UCQs, resulting in a rather detailed classification. We present it in this section as a reference point and as a basis for establishing our own classifications later on. We start with introducing the various structural measures that play a role in the classification.

Let $q(\bar{x}) = \exists \bar{y}\, \varphi(\bar{x}, \bar{y})$ be a CQ. The *Gaifman graph* of q, denoted G_q, is G_{D_p} where CQ p is obtained from q by replacing answer variable x_2 with answer variable x_1 whenever $x_1 = x_2$ is an atom in q. The *treewidth (TW)* of $q(\bar{x})$ is the treewidth of G_q.

An $\bar{x}$-*component* of G_q is the undirected graph obtained as follows: (1) take the subgraph of G_q induced by vertex set $\mathsf{var}(q) \setminus \bar{x}$, (2) choose a maximal connected component (V_c, E_c), and (3) re-add all edges from G_q that contain at least one vertex from V_c. The *contract* of G_q, denoted $\mathsf{contract}(G_q)$, is the restriction of G_q to the variables in $\bar{x}$, extended with every edge $\{x_1, x_2\} \subseteq \bar{x}$ such that x_1, x_2 co-occur in some $\bar{x}$-component of G_q. We shall often be interested in the treewidth of the contract of a CQ q, which we refer to as the *contract treewidth (CTW)* of q.

The *starsize (SS)* of q is the maximum number of answer variables in any $\bar{x}$-component of G_q. Note that the same notion is called strict starsize in [16] and dominating starsize in [19]. It is different from the original notion of starsize from [20, 21].

A set of quantified variables S in q is *node-well-linked* if for every two disjoint sets $S_1, S_2 \subseteq S$ of the same cardinality, there are $|S_1|$ vertex disjoint paths in G_q that connect the vertices in S_1 with the vertices in S_2. For example, S is node-well-linked if $D_q|_S$ takes the form of a grid or of a clique. A matching M from the answer variables $\bar{x}$ to the quantified



(a) TW = 3, CTW = 1. **(b)** CTW = 3, SS = 2. **(c)** SS = 4, LMN = 1. **(d)** LMN = 4.

Figure 1 Examples for structural measures: Example (a) is the (3,3)-complete bipartite graph, the contract of Example (b) is the 4-clique, Example (c) is a 4-star, and Example (d) is a 4-star with the 4-clique in the centre. Filled nodes are answer variables, hollow nodes are quantified variables.

variables $\mathsf{var}(q) \setminus \bar{x}$ in the graph G_q (in the standard sense of graph theory) is *linked* if the set S of quantified variables that occur in M is node-well-linked. The *linked matching number (LMN)* of q is the size of the largest linked matching from $\bar{x}$ to $\mathsf{var}(q) \setminus \bar{x}$ in G_q. One should think of the linked matching number as a strengthening of starsize. We do not only demand that many answer variables are interlinked by the same $\bar{x}$-component, but additionally require that this component is sufficiently large and highly connected ("linked").

Figure 1 contains some example CQs with associated measures. For a class of CQs $\mathbb{C}$, the contract treewidths of CQs in $\mathbb{C}$ being bounded by a constant implies that the same is true for starsizes, and bounded starsizes in turn imply bounded linked matching numbers. There are no implications between bounded treewidths and bounded contract treewidths; in Figure 1, Example (a) generalizes to any treewidth while always having contract treewidth 1 and Example (c), which has contract treewidth 3, generalizes to any contract treewidth (and starsize) while always having treewidth 1. See also [16, 19] for additional examples.

It is a fundamental observation that cores of CQs are guaranteed to have minimum measures among all equivalent CQs, as stated by the following lemma [16, 19].

▶ **Lemma 2.** *If a CQ q is equivalent to a CQ of treewidth k, then $\mathsf{core}(q)$ has treewidth at most k. The same is true for contract treewidth, starsize, and linked matching number.*

An additional ingredient needed to formulate the classification for UCQs emerges from [17]. There, Chen and Mengel associate with every UCQ q a set of CQs $\mathsf{cl}_{\mathsf{CM}}(q)$ such that, informally speaking, counting the number of answers to q is equivalent to counting the number of answers to the CQs in $\mathsf{cl}_{\mathsf{CM}}(q)$. We now introduce this set in detail.

Two CQs $q_1(\bar{x}_1)$ and $q_2(\bar{x}_2)$ over the same schema $\mathbf{S}$ are *counting equivalent* if $\#q_1(D) = \#q_2(D)$ for all $\mathbf{S}$-databases D. Let $q(\bar{x}) = p_1 \vee \cdots \vee p_n$. The starting point for defining $\mathsf{cl}_{\mathsf{CM}}(q)$ is the observation that, by the inclusion-exclusion principle, every database D satisfies

$$\#q(D) \quad = \quad \sum_{I \subseteq [n]} (-1)^{|I|+1} \cdot \#\Big(\bigwedge_{i \in I} p_i(D) \Big)$$

We can manipulate this sum as follows: if there are two summands $c_1 \cdot \#\big(\bigwedge_{i \in I_1} p_i(D) \big)$ and $c_2 \cdot \#\big(\bigwedge_{i \in I_2} p_i(D) \big)$ such that $\bigwedge_{i \in I_1} p_i$ and $\bigwedge_{i \in I_2} p_i$ are counting equivalent, then delete both summands and add $(c_1 + c_2) \cdot \#\big(\bigwedge_{i \in I_1} p_i(D) \big)$ to the sum. After doing this exhaustively, delete all summands with coefficient zero. The elements of $\mathsf{cl}_{\mathsf{CM}}(q)$ are all CQs $\bigwedge_{i \in I} p_i$ in the original sum that are counting equivalent to some CQ CQ $\bigwedge_{i \in J} p_i$ which remains in

the sum.[2] Note that the number of CQs in $\mathsf{cl}_{\mathsf{CM}}(q)$ might be exponentially larger than the number of CQs in q and that $\mathsf{cl}_{\mathsf{CM}}(q)$ does not need to contain all CQs from the original UCQ q. The main property of $\mathsf{cl}_{\mathsf{CM}}(q)$ is as follows.

▶ **Lemma 3** ([17]). *Let q be a UCQ over schema $\mathbf{S}$ and D an $\mathbf{S}$-database. Then $\#q(D)$ can be computed in polynomial time from the counts $\#q'(D)$, $q' \in \mathsf{cl}_{\mathsf{CM}}(q)$. Conversely, for every $q' \in \mathsf{cl}_{\mathsf{CM}}(q)$, there is a set of databases $D_1, \ldots, D_n$ such that $\#q'(D)$ can be computed in polynomial time from the counts $\#q(D_i)$, $1 \leq i \leq n$, and $D_1, \ldots, D_n$ can be computed in time $f(||q||) \cdot p(||D||)$ where f is a computable function and p is a polynomial.*

For the first part of Lemma 3, note that $\#q(D)$ can be computed from $\#q'(D)$, $q' \in \mathsf{cl}_{\mathsf{CM}}(q)$, simply by evaluating the sum derived above from the inclusion-exclusion principle, after the manipulation.

We are now ready to state the characterization.

▶ **Theorem 4** ([16, 17, 19]). *Let $\mathbb{Q} \subseteq \mathbb{UCQ}$ be recursively enumerable and have relation symbols of bounded arity, and let $\mathbb{Q}^\star = \{\mathsf{core}(q) \mid q \in \mathsf{cl}_{\mathsf{CM}}(\mathbb{Q})\}$. Then the following holds:*

1. *If the treewidths and the contract treewidths of CQs in $\mathbb{Q}^\star$ are bounded, then $\mathsf{AnswerCount}(\mathbb{Q})$ is in FPT; it is even in* PTime *when $\mathbb{Q} \subseteq \mathbb{CQ}$.*
2. *If the treewidths of CQs in $\mathbb{Q}^\star$ are unbounded and the contract treewidths of CQs in $\mathbb{Q}^\star$ are bounded, then $\mathsf{AnswerCount}(\mathbb{Q})$ is $\mathrm{W}[1]$-equivalent.*
3. *If the contract treewidths of CQs in $\mathbb{Q}^\star$ are unbounded and the starsizes of CQs in $\mathbb{Q}^\star$ are bounded, then $\mathsf{AnswerCount}(\mathbb{Q})$ is $\#\mathrm{W}[1]$-equivalent.*
4. *If the starsizes of CQs in $\mathbb{Q}^\star$ are unbounded, then $\mathsf{AnswerCount}(\mathbb{Q})$ is $\#\mathrm{W}[2]$-hard.*
5. *If the linked matching numbers of CQs in $\mathbb{Q}^\star$ are unbounded, then $\mathsf{AnswerCount}(\mathbb{Q})$ is $\#\mathrm{A}[2]$-equivalent.*

We remark that $\mathsf{cl}_{\mathsf{CM}}(q) = \{q\}$ when q is a CQ, and thus it suffices to define $\mathbb{Q}^\star$ as $\{\mathsf{core}(q) \mid q \in \mathbb{Q}\}$ when $\mathbb{Q} \subseteq \mathbb{CQ}$ in Theorem 4.

Note that the classification given by Theorem 4 is not complete. It leaves open the possibility that there is a class of (U)CQs $\mathbb{Q}$ such that $\mathsf{AnswerCount}(\mathbb{Q})$ is $\#\mathrm{W}[2]$-hard, but neither $\#\mathrm{W}[2]$-equivalent nor $\#\mathrm{A}[2]$-equivalent. It is conjectured in [19] that such a class $\mathbb{Q}$ indeed exists and in particular that there might be classes $\mathbb{Q}$ such that $\mathsf{AnswerCount}(\mathbb{Q})$ is $\#\mathrm{W}_{\mathsf{func}}[2]$-equivalent. The classification also leaves open whether unbounded linked matching numbers is a necessary condition for $\#\mathrm{A}[2]$-hardness. While a complete classification is certainly desirable we note that, from our perspective, the most relevant aspect is the delineation of the FPT cases from the hard cases, achieved by Points 1-3 of the theorem.

4 Problems Studied and Main Results

We introduce the problems studied and state the main results of this paper. We start with ontology-mediated querying and then proceed to querying under constraints.

An *ontology* $\mathcal{O}$ is a finite set of TGDs. An *ontology mediated query (OMQ)* takes the form $Q = (\mathcal{O}, \mathbf{S}, q)$ where $\mathcal{O}$ is an ontology, $\mathbf{S}$ is a finite schema called the *data schema*, and q is a UCQ. Both $\mathcal{O}$ and q can use symbols from $\mathbf{S}$, but also additional symbols, and in particular $\mathcal{O}$ can "introduce" additional symbols to enrich the vocabulary available for querying. We assume w.l.o.g. that all relation symbols in q that are not from $\mathbf{S}$ occur also in $\mathcal{O}$. This can

[2] This definition slightly deviates from that of Chen and Mengel, who include no two CQs that are counting equivalent. For all relevant purposes, however, the two definitions are interchangable.

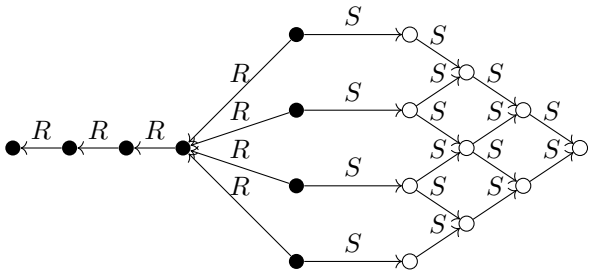


Figure 2 CQ q_4 from Example 5. Filled circles indicate answer variables.

always be achieved by introducing dummy TGDs $R(\bar{x}) \to R(\bar{x})$. When $\mathcal{O}$ and q only use symbols from $\mathbf{S}$, then we say that the data schema of Q is *full*. The *arity* of Q is defined as the arity of q. We write $Q(\bar{x})$ to emphasize that the answer variables of q are $\bar{x}$ and for brevity often refer to the data schema simply as the schema.

A tuple $\bar{c} \in \mathsf{adom}(D)^{|\bar{x}|}$ is an *answer* to Q over D if $\bar{c} \in q(I)$ for each model I of $\mathcal{O}$ with $I \supseteq D$. The *evaluation of* $Q(\bar{x})$ over D, denoted $Q(D)$, is the set of all answers to Q over D. Note that, as a consequence of Lemma 1, $Q(D) = q(\mathsf{ch}_{\mathcal{O}}(D))$ for every OMQ $Q = (\mathcal{O}, \mathbf{S}, q)$ and $\mathbf{S}$-database D.

An *OMQ language* is a class of OMQs. For a class of TGDs $\mathbb{C}$ and a class of UCQs $\mathbb{Q}$, we write $(\mathbb{C}, \mathbb{Q})$ to denote the OMQ language that consists of all OMQs $(\mathcal{O}, \mathbf{S}, q)$ where $\mathcal{O}$ is a set of TGDs from $\mathbb{C}$ and $q \in \mathbb{Q}$. For example, we may write $(\mathbb{G} \cap \mathbb{FULL}, \mathbb{UCQ})$. We say that an OMQ language $(\mathbb{C}, \mathbb{Q})$ has *full data schema* if every OMQ in it has. If $\mathbb{Q}$ is an OMQ language, the problem $\mathsf{AnswerCount}(\mathbb{Q})$ is defined exactly as in Section 2 with query language $\mathbb{Q}$.

Our first main result is a counterpart of Theorem 4 for OMQs from $(\mathbb{G}, \mathbb{UCQ})$ based on the full schema. To illustrate the effect of adding an ontology, we first observe that the ontology interacts with all of the measures in Theorem 4.

▶ **Example 5.** Let $\mathcal{O} = \{R(x,y) \to S(x,y)\}$ and $\mathbf{S} = \{R, S\}$. For all $n \geq 0$, let

$$q_n(x_1, \dots, x_n, z_1, \dots, z_n) \quad = \quad \exists_{1<i+j<n+2}\, y_{i,j} \bigwedge_{1\leq i\leq n} R(x_i, z_1) \wedge \bigwedge_{1\leq i<n} R(z_i, z_{i+1}) \wedge \bigwedge_{i+j=n+1} S(x_i, y_{i,j}) \wedge \bigwedge_{2<i+j<n+2} S(y_{i+1,j}, y_{i,j}) \wedge S(y_{i,j+1}, y_{i,j}).$$

Then q_n is a core of treewidth $\lfloor \frac{n}{2} \rfloor$, contract treewidth n, starsize n, and linked matching number n. But the OMQ $(\mathcal{O}, \mathbf{S}, q_n)$ is equivalent to $(\mathcal{O}, \mathbf{S}, p_n)$ with p_n obtained from q_n by dropping all S-atoms. Since p_n is tree-shaped and has no quantified variables, all measures are at most 1. Figure 2 depicts query q_4.

Before we state our characterization, we observe as a preliminary that OMQs from $(\mathbb{G}, \mathbb{UCQ})$ can be rewritten into equivalent ones from $(\mathbb{G} \cap \mathbb{FULL}, \mathbb{UCQ})$, that is, existential quantifiers can be removed from rule heads when the actual query is adjusted in a suitable way. This has already been observed, for example, in [5].

▶ **Theorem 6.** *For every OMQ $Q \in (\mathbb{G}, \mathbb{UCQ})$, there is an equivalent OMQ from $(\mathbb{G} \cap \mathbb{FULL}, \mathbb{UCQ})$ that can be effectively computed.*

The proof of Theorem 6 is constructive, that is, it provides an explicit way of computing, given an OMQ $Q = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$, an equivalent OMQ from $(\mathbb{G} \cap \mathbb{FULL}, \mathbb{UCQ})$. We denote this OMQ with $Q^{\exists} = (\mathcal{O}^{\exists}, \mathbf{S}, q^{\exists})$ and call it the $\exists$-*rewriting* of Q. It is worth

noting that even if q contains no equality atoms, such atoms might be introduced during the construction of $q^\exists$. This is the main reason for admitting equality atoms in (U)CQs in this paper in the first place.

For OMQs $Q \in (\mathbb{G}, \mathbb{UCQ})$, we define a set $\mathsf{cl}_{\mathsf{CM}}(Q)$ in exact analogy with the definition of $\mathsf{cl}_{\mathsf{CM}}(q)$ for UCQs q, that is, for $Q = (\mathcal{O}, \mathbf{S}, p_1 \vee \cdots \vee p_n)$, we use the OMQs $(\mathcal{O}, \mathbf{S}, p_i)$ in place of the CQs p_i from the UCQ q in the definition of $\mathsf{cl}_{\mathsf{CM}}(Q)$. This requires the use of counting equivalence for OMQs, which is as defined in the expected way. For a class $\mathbb{Q} \subseteq (\mathbb{G}, \mathbb{UCQ})$, we now identify a class $\mathbb{Q}^\star$ of CQs by setting

$$\mathbb{Q}^\star = \{\mathsf{core}(\mathsf{ch}_{\mathcal{O}^\exists}(p)) \mid \exists Q \in \mathbb{C} : (\mathcal{O}^\exists, \mathbf{S}, p) \in \mathsf{cl}_{\mathsf{CM}}(Q^\exists)\}.$$

Our main result is now as follows.

▶ **Theorem 7.** *Let $\mathbb{Q} \subseteq (\mathbb{G}, \mathbb{UCQ})$ be a recursively enumerable class of OMQs with full data schema and relation symbols of bounded arity. Then the following hold:*

1. *If the treewidths and contract treewidths of CQs in $\mathbb{Q}^\star$ are bounded, then* $\mathsf{AnswerCount}(\mathbb{Q})$ *is in FPT.*
2. *If the treewidths of CQs in $\mathbb{Q}^\star$ are unbounded and the contract treewidths of CQs in $\mathbb{Q}^\star$ are bounded, then* $\mathsf{AnswerCount}(\mathbb{Q})$ *is* $\#\mathrm{W}[1]$*-equivalent.*
3. *If the contract treewidths of CQs in $\mathbb{Q}^\star$ are unbounded and the starsizes of CQs in $\mathbb{Q}^\star$ are bounded, then* $\mathsf{AnswerCount}(\mathbb{Q})$ *is* $\#\mathrm{W}[1]$*-equivalent.*
4. *If the starsizes of CQs in $\mathbb{Q}^\star$ are unbounded, then* $\mathsf{AnswerCount}(\mathbb{Q})$ *is* $\#\mathrm{W}[2]$*-hard.*
5. *If the linked matching numbers of CQs in $\mathbb{Q}^\star$ are unbounded, then* $\mathsf{AnswerCount}(\mathbb{Q})$ *is* $\#\mathrm{A}[2]$*-equivalent.*

Points 1 to 5 of Theorem 7 parallel exactly those of Theorem 4, but of course the definition of $\mathbb{Q}^\star$ is a different one. It is through this definition that we capture the potential interaction between the ontology and the structural measures. Note, for example, that the class of OMQs $(\mathcal{O}, \mathbf{S}, q_n)$, $n \geq 1$, from Example 5 would be classified as $\#\mathrm{A}[2]$-equivalent if $\mathsf{core}(\mathsf{ch}_{\mathcal{O}^\exists}(p))$ was replaced with p in the definition of $\mathbb{Q}^\star$ while it is in fact in FPT. Also note that the PTime statement in Point 1 of Theorem 4 is absent in Theorem 7. In fact, evaluating Boolean OMQs from $(\mathbb{G}, \mathbb{UCQ})$ is 2ExpTime-complete [12] and since for Boolean OMQs evaluation coincides with answer counting, PTime cannot be attained.

Our second main result concerns querying under integrity constraints that take the form of guarded TGDs. In contrast to OMQs, the constraints are thus not used for deductive reasoning, but instead give rise to a promise regarding the shape of the input database. Following [5], we define a *constraint-query specification* (CQS) as a triple $S = (\mathcal{T}, \mathbf{S}, q)$ where $\mathcal{T}$ is a set of TGDs over finite schema $\mathbf{S}$ and q a UCQ over $\mathbf{S}$. We call $\mathcal{T}$ the set of *integrity constraints*. Overloading notation, we write $(\mathbb{C}, \mathbb{Q})$ for the class of CQSs in which the set of integrity constraints is formulated in the class of TGDs $\mathbb{C}$, and the query is coming from the class of queries $\mathbb{Q}$. It will be clear from the context whether $(\mathbb{C}, \mathbb{Q})$ is an OMQ language or a class of CQSs. Every class $\mathbb{C}$ of CQSs gives rise to the following answer counting problem.

PROBLEM :	$\mathsf{AnswerCount}(\mathbb{C})$
INPUT :	A set of TGDs $\mathcal{T}$, a query q, and an $\mathbf{S}$-database D that satisfies $\mathcal{T}$ such that $(\mathcal{T}, \mathbf{S}, q) \in \mathbb{C}$.
OUTPUT :	$\#q(D)$

Our second main result parallels Theorems 4 and 7. We refrain from explicitly listing all cases again.

▶ **Theorem 8.** *Let $\mathbb{Q} \subseteq (\mathbb{G}, \mathbb{UCQ})$ be a recursively enumerable class of CQSs with relation symbols of bounded arity. Then Statements 1-5 of Theorem 7 hold.*

Note that the delineation of the considered complexities is identical for ontology-mediated querying and for querying under constraints. In particular, Theorem 8 (implicitly) uses exactly the same class of CQs $\mathbb{Q}^\star$ and the same associated measures.

It would be interesting to know whether $\mathsf{AnswerCount}(\mathbb{Q})$ being in FPT coincides with $\mathsf{AnswerCount}(\mathbb{Q})$ being in PTime for classes of CQSs $\mathbb{Q} \subseteq (\mathbb{G}, \mathbb{CQ})$. Note that this is the case for evaluation in the presence of constraints that are guarded TGDs [8, 7] and also for answer counting without constraints [16]. The proof of these results, however, break in our setting.

We derive Theorem 8 from Theorem 7 by means of reduction. In fact, Theorem 8 is a consequence of Theorem 7 and the following result.

▶ **Theorem 9.** *Let $\mathbb{C} \subseteq (\mathbb{G}, \mathbb{UCQ})$ be a recursively enumerable class of CQSs and let $\mathbb{C}'$ be $\mathbb{C}$ viewed as a class of OMQs based on the full schema.*[3] *Then there is a Turing fpt-reduction from $\mathsf{AnswerCount}(\mathbb{C}')$ to $\mathsf{AnswerCount}(\mathbb{C})$ and there is a parsimonious polynomial time reduction from $\mathsf{AnswerCount}(\mathbb{C})$ to $\mathsf{AnswerCount}(\mathbb{C}')$.*

The reduction from $\mathsf{AnswerCount}(\mathbb{C})$ to $\mathsf{AnswerCount}(\mathbb{C}')$ is immediate: given a set of guarded TGDs $\mathcal{T}$, a CQ q, and an $\mathbf{S}$-database D that satisfies $\mathcal{T}$, we can view $(\mathcal{T}, \mathbf{S}, q)$ as an OMQ Q based on the full schema and return $\#Q(D)$ as $\#q(D)$. It is easy to see that this is correct.

For the converse reduction, we are given a $Q = (\mathcal{O}, \mathbf{S}, q)$ that is a CQS from $\mathbb{C}$ viewed as an OMQ and an $\mathbf{S}$-database D. It seems a natural idea to simply view Q as a CQS, which it originally was, and replace D with $\mathsf{ch}_\mathcal{O}(D)$ so that the promise is satisfied, and to then return $\#q(\mathsf{ch}_\mathcal{O}(D))$ as $\#Q(D)$. However, there are two obstacles. First, $\mathsf{ch}_\mathcal{O}(D)$ need not be finite; and second, chasing adds fresh constants which changes the answer count. We solve the first problem by replacing the infinite chase with a (finite!) database $D^\star$ that extends D and satisfies $\mathcal{O}$. The following result from [5] is essentially a consequence of $\mathbb{G}$ being finitely controllable.

▶ **Theorem 10** ([5])**.** *Given an ontology $\mathcal{O} \subseteq \mathbb{G}$, an $\mathbf{S}$-database D, and an $n \geq 1$, one can effectively construct a finite database D^* that satisfies the following conditions:*

1. *$D^* \models \mathcal{O}$ and $D \subseteq D^*$;*
2. *$\bar{a} \in q(D^*)$ iff $\bar{a} \in Q(D)$ for all OMQs $(\mathcal{O}, \mathbf{S}, q)$ where q has at most n variables and for all tuples $\bar{a}$ that use only constants in $\mathsf{adom}(D)$.*

The construction of D^* takes time $||D||^{O(1)} \cdot f(||\mathcal{O}|| + n)$.

To address the second problem, we correct the count. Note that this cannot be done by introducing fresh unary relation symbols as markers to distinguish the original constants from those introduced by the chase as this would require us to change the query. We instead use an approach inspired by [16]. The idea is to compute $\#q(D')$ on a set of databases D' obtained from $D^\star$ by cloning constants in $\mathsf{adom}(D) \subseteq \mathsf{adom}(D^\star)$. The results can be arranged in a system of equations whose coefficients form a Vandermonde matrix. Finally, the system can be solved to obtain $\#q(D)$. This is formalized by the following lemma where we use $\mathsf{clones}(D)$ to denote the class of all $\mathbf{S}$-databases that can be obtained from $\mathbf{S}$-database D by cloning constants. A proof is in the appendix of the long version.

[3] Syntactically, a CQS $(\mathcal{T}, \mathbf{S}, q)$ and an OMQ $(\mathcal{T}, \mathbf{S}, q)$ are actually the same thing except that the definition of CQSs is more strict regarding the schema $\mathbf{S}$; as a consequence when viewing a CQS as an OMQ, the latter is based on the full schema.

▶ **Lemma 11.** *Fix a constant r. There is a polynomial time algorithm that, given a UCQ $q(\bar{x})$ over schema $\mathbf{S}$ with $\mathsf{ar}(\mathbf{S}) \leq r$, an $\mathbf{S}$-database D, and a set $F \subseteq \mathsf{adom}(D)$, computes $\#(q(D) \cap F^{|\bar{x}|})$ using an oracle for $\mathsf{AnswerCount}(\{q\}, \mathsf{clones}(D))$.*

Now for the reduction from $\mathsf{AnswerCount}(\mathbb{C}')$ to $\mathsf{AnswerCount}(\mathbb{C})$ claimed in Theorem 9. Let $Q(\bar{x}) = (\mathcal{O}, S, q)$ be a CQS from $\mathbb{C}$ viewed as an OMQ, and let D be an $\mathbf{S}$-database. We first construct the database D^* as per Theorem 10 with n being the number of variables in q. We then apply the algorithm asserted by Lemma 11 with D^* in place of D and with $F := \mathsf{adom}(D)$. Cloning preserves guarded TGDs and thus we can use the oracle (which can compute $\#q(D')$ for any $\mathbf{S}$-database D' that satisfies $\mathcal{O}$) for computing $\mathsf{AnswerCount}(\{q\}, \mathsf{clones}(D))$ as required by Lemma 11.

5 Proof of Theorem 7

We first establish the upper bounds in Theorem 7, starting with the FPT upper bound from Point 1. Let $\mathbb{C} \subseteq (\mathbb{G}, \mathbb{UCQ})$ be a class of OMQs such that the treewidths and the contract treewidths of CQs in $\mathbb{Q}^\star$ are bounded by a constant k. Given an OMQ $Q \in \mathbb{C}$ and an $\mathbf{S}$-database D, we first replace Q by its $\exists$-rewriting $Q^\exists = (\mathcal{O}^\exists, \mathbf{S}, q^\exists)$. Since Q is equivalent to $Q^\exists$ we have that $\#Q(D) = \#Q^\exists(D)$, thus it suffices to compute the latter count. The first part of Lemma 3 clearly lifts from UCQs to OMQs, see also the remark after that lemma. We can thus compute $\#Q^\exists(D)$ (essentially by applying the inclusion-exclusion principle) within the time requirements of FPT once we have computed $\#(\mathcal{O}^\exists, \mathbf{S}, p)(D)$ for all p such that $(\mathcal{O}^\exists, \mathbf{S}, p) \in \mathsf{cl}_{\mathsf{CM}}(Q^\exists)$. By the universality of the chase, $\#(\mathcal{O}^\exists, \mathbf{S}, p)(D) = \#p(\mathsf{ch}_{\mathcal{O}^\exists}(D))$. Moreover, since $\mathcal{O}^\exists$ is from $\mathbb{G} \cap \mathbb{FULL}$, $\mathsf{ch}_{\mathcal{O}^\exists}(D)$ is finite and can be computed within the time requirements of FPT.

Thus, to compute $\#Q(D)$ it is enough to compute $\#p(\mathsf{ch}_{\mathcal{O}^\exists}(D))$ for all CQs p such that $(\mathcal{O}^\exists, \mathbf{S}, p) \in \mathsf{cl}_{\mathsf{CM}}(Q^\exists)$ or, equivalently, to compute $\#\mathsf{core}(\mathsf{ch}_{\mathcal{O}^\exists}(p))(\mathsf{ch}_{\mathcal{O}^\exists}(D))$ for all p with $(\mathcal{O}^\exists, \mathbf{S}, p) \in \mathsf{cl}_{\mathsf{CM}}(Q^\exists)$. But the CQs $\mathsf{core}(\mathsf{ch}_{\mathcal{O}^\exists}(p))$ for these p are exactly the CQs from $\mathbb{Q}^\star$ and thus their treewidths and contract treewidths are bounded by k. Consequently, we can apply the fpt algorithm from Point 1 of Theorem 4 as a black box and overall obtain an FPT procedure. The remaining upper bounds from Theorem 7 can be proved analogously, exploiting that easiness for W[1] and #W[1] is defined in terms of Turing fpt-reductions.

We next turn towards lower bounds, which are proved by a sequence of Turing fpt-reductions. The first such reduction consists in transitioning to the $\exists$-rewritings of the OMQs in the original class. The second reduction enables us to consider OMQs that use CQs rather than UCQs.[4] And in the third reduction, we remove ontologies altogether, that is, we reduce classes of CQs to classes of OMQs. We start with the first reduction.

▶ **Theorem 12.** *Let $\mathbb{C} \subseteq (\mathbb{G}, \mathbb{UCQ})$ be recursively enumerable and let $\mathbb{C}' \subseteq (\mathbb{G} \cap \mathbb{FULL}, \mathbb{UCQ})$ be the class of $\exists$-rewritings of OMQs from $\mathbb{C}$. There is a parsimonious fpt-reduction from $\mathsf{AnswerCount}(\mathbb{C}')$ to $\mathsf{AnswerCount}(\mathbb{C})$.*

Proof. Given a $Q = (\mathcal{O}, \mathbf{S}, q) \in \mathbb{C}'$ and an $\mathbf{S}$-database D, find some $Q' = (\mathcal{O}', \mathbf{S}, q') \in \mathbb{C}$ such that Q is an $\exists$-rewriting of Q' by recursively enumerating $\mathbb{C}'$ and exploiting that OMQ equivalence is decidable in $(\mathbb{G}, \mathbb{UCQ})$ [4], then compute and return $\#Q'(D)$. ◀

The second reduction is given by the following theorem.

[4] The construction of $Q^\exists$ may produce a UCQ even if the original OMQ contained a CQ.

▶ **Theorem 13.** *Let $\mathbb{C} \subseteq (\mathbb{G} \cap \mathbb{FULL}, \mathbb{UCQ})$ be a recursively enumerable class of OMQs with full schema and relation symbols of bounded arity, and let $\mathbb{C}' \subseteq (\mathbb{G} \cap \mathbb{FULL}, \mathbb{CQ})$ be the class of OMQs $\{Q' \mid \exists Q \in \mathbb{C} : Q' \in \mathsf{cl}_{\mathsf{CM}}(Q)\}$. Then there is a Turing fpt-reduction from* $\mathsf{AnswerCount}(\mathbb{C}')$ *to* $\mathsf{AnswerCount}(\mathbb{C})$.

In [17], Chen and Mengel establish Theorem 13 in the special case where ontologies are empty. However, a careful analysis reveals that their proof extends to recursively enumerable classes $\mathbb{D}$ of databases over some schema $\mathbf{S}$ that satisfy the following conditions:

1. $\mathbb{D}$ is closed under disjoint unions, direct products, and contains the *well of positivity* $D^\top_{\mathbf{S}}$, that is, the $\mathbf{S}$-database with a single constant c defined as $D^\top_{\mathbf{S}} = \{R(c, \dots, c) \mid R \in \mathbf{S}\}$;
2. counting equivalence and semi-counting equivalence between CQs over $\mathbb{D}$ is decidable, where CQs $q_1(\bar{x}_1)$ and $q_2(\bar{x}_2)$ over the same schema $\mathbf{S}$ are *semi-counting equivalent* if they are counting equivalent over all $\mathbf{S}$-databases D with $\#q_1(D) > 0$ and $\#q_2(D) > 0$.

This allows us to establish Theorem 13 by observing that models of TGDs are closed under the operations mentioned in Point 1and that the two equivalence problems in Point 2 are both decidable. Details are in the appendix of the long version.

We next give the reduction that removes ontologies.

▶ **Theorem 14.** *Let $\mathbb{C} \subseteq (\mathbb{G} \cap \mathbb{FULL}, \mathbb{CQ})$ be a recursively enumerable class of OMQs with full schema and relation symbols of bounded arity. There is a class $\mathbb{C}' \subseteq \mathbb{CQ}$ that only contains cores and such that:*

1. *there is a Turing fpt-reduction from* $\mathsf{AnswerCount}(\mathbb{C}')$ *to* $\mathsf{AnswerCount}(\mathbb{C})$*;*
2. *for every OMQ $Q = (\mathcal{O}, \mathbf{S}, q) \in \mathbb{C}$, we find a CQ $p \in \mathbb{C}'$ such that the treewidth of p is equal to that of* $\mathsf{core}(\mathsf{ch}_{\mathcal{O}}(q))$*, and likewise for contract treewidth, starsize, and linked matching number.*

Using Theorems 12, 13, and 14, we can make use of the lower bounds for classes of (U)CQs stated in Theorem 4 to prove the lower bounds in Theorem 7. Let us consider, for example, the W[1] lower bound from Point 2. Take a class $\mathbb{C}_0 \subseteq (\mathbb{G}, \mathbb{UCQ})$ of OMQs such that the treewidths of CQs in

$$\mathbb{Q}^\star = \{\mathsf{core}(\mathsf{ch}_{\mathcal{O}^\exists}(p)) \mid \exists Q \in \mathbb{C}_0 : (\mathcal{O}^\exists, \mathbf{S}, p) \in \mathsf{cl}_{\mathsf{CM}}(Q^\exists)\}$$

are unbounded. Theorems 12 and 13 give a Turing fpt-reduction from $\mathsf{AnswerCount}(\mathbb{C})$ to $\mathsf{AnswerCount}(\mathbb{C}_0)$ where

$$\mathbb{C} = \{Q' \mid \exists Q \in \mathbb{C}_0 : Q' \in \mathsf{cl}_{\mathsf{CM}}(Q^\exists)\}.$$

By assumption, the treewidths of the CQs $\mathsf{core}(\mathsf{ch}_{\mathcal{O}}(q))$, $(\mathcal{O}, \mathbf{S}, q) \in \mathbb{C}$, are unbounded. Theorem 14 thus yields a Turing fpt-reduction from $\mathsf{AnswerCount}(\mathbb{C}')$ to $\mathsf{AnswerCount}(\mathbb{C})$ for some class of CQs $\mathbb{C}'$ that are all cores and such that the treewidths of CQs in $\mathbb{C}'$ are unbounded. By Point 2 of Theorem 4, $\mathsf{AnswerCount}(\mathbb{C}')$ is W[1]-hard. Composing the reductions, we obtain a Turing fpt-reduction from $\mathsf{AnswerCount}(\mathbb{C}')$ to $\mathsf{AnswerCount}(\mathbb{C}_0)$. The other lower bounds can be proved analogously.

Now for the proof of Theorem 14. It in turn uses two consecutive fpt-reductions. In the first step, we show that we can assume that every variable in a CQ (inside an OMQ) is marked by a unary relation symbol that identifies the variable. The *marking* of a CQ q over schema $\mathbf{S}$ is the CQ q^m obtained from q by adding the atom $R_x(x)$, for each $x \in \mathsf{var}(q)$ and with R_x a fresh unary relation symbol. Note that q^m is over schema $\mathbf{S}^m$ obtained from $\mathbf{S}$ by adding all the fresh symbols. The *core-chased marking* of an OMQ $Q = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{CQ})$ is the OMQ $Q^m = (\mathcal{O}, \mathbf{S}^m, \mathsf{core}(\mathsf{ch}_{\mathcal{O}}(q))^m) \in (\mathbb{G}, \mathbb{CQ})$. We lift this to classes of OMQs $\mathbb{C}$ as expected, that is, $\mathbb{C}^m = \{Q^m \mid Q \in \mathbb{C}\}$.

▶ **Lemma 15.** *Let $\mathbb{C} \subseteq (\mathbb{G} \cap \mathbb{FULL}, \mathbb{CQ})$ be a recursively enumerable class of OMQs with full schema and relation symbols of bounded arity. Then there is a Turing fpt-reduction from* $\mathsf{AnswerCount}(\mathbb{C}^m)$ *to* $\mathsf{AnswerCount}(\mathbb{C})$.

The proof of Lemma 15 in the appendix of the long version lifts a corresponding proof from [16] that applies to CQs without ontologies, essentially by verifying that the proof also applies to classes of databases chased with an ontology from $\mathbb{G} \cap \mathbb{FULL}$. We next observe that in the presence of markings it is possible to get rid of ontologies, in the following sense.

▶ **Lemma 16.** *Let $\mathbb{C}^m \subseteq (\mathbb{G} \cap \mathbb{FULL}, \mathbb{CQ})$ be a recursively enumerable class of OMQs with full schema and relation symbols of bounded arity that are core-chased markings. There exists a class $\mathbb{C} \subseteq \mathbb{CQ}$ of cores with relation symbols of bounded arity such that:*

1. *there is a Turing fpt-reduction from* $\mathsf{AnswerCount}(\mathbb{C})$ *to* $\mathsf{AnswerCount}(\mathbb{C}^m)$*;*
2. *$\mathbb{C}$ is based on the same Gaifman graphs as $\mathbb{C}^m$: $\{G_q \mid q \in \mathbb{C}^m\} = \{G_q \mid (\mathcal{O}, \mathbf{S}, q) \in \mathbb{C}\}$.*

We provide a proof of Lemma 16 below. Before, however, we show how Theorem 14 follows from Lemmas 15 and 16.

Proof of Theorem 14. Let $\mathbb{C} \subseteq (\mathbb{G} \cap \mathbb{FULL}, \mathbb{CQ})$ be a recursively enumerable class of OMQs with full schema and relation symbols of bounded arity. From Lemma 16, we obtain a class $\mathbb{C}'$ of CQs that are cores and are based on the same Gaifman graphs as $\mathbb{C}^m$. This is the class whose existence is postulated by Theorem 14. We briefly argue that Points 1 and 2 of that theorem are satisfied. The Turing fpt-reduction required by Point 1 is the composition of the reductions asserted by Lemmas 15 and 16. Point 2 is a consequence of the facts that (1) the structural measures of a CQ are defined through its Gaifman graph, (2) $\mathbb{C}'$ is based on the same Gaifman graphs as $\mathbb{C}^m$, and (3) marking a CQ does not affect its Gaifman graph. ◀

Now for the announced proof of Lemma 16, a key ingredient to the proof of Theorem 7.

Proof of Lemma 16. To prove the lemma, we define the required class of CQs $\mathbb{C}$ and describe an fpt algorithm that

- takes as an input a query $q \in \mathbb{C}$ over schema $\mathbf{S}$ and an $\mathbf{S}$-database D,
- has access to an oracle for $\mathsf{AnswerCount}(\mathbb{C}^m)$, and
- outputs $\#q(D)$.

A guarded set in a database D is a set $S \subseteq \mathsf{adom}(D)$ such that all constants in S jointly occur in a fact in D, possibly together with additional constants. With a maximal guarded set, we mean a guarded set that is maximal regarding set inclusion.

The class $\mathbb{C}$ contains one CQ q^s for every OMQ $Q = (\mathcal{O}, \mathbf{S}^m, q^m) \in \mathbb{C}$ that is formulated in a different schema introduced below (whence the superscript "s"). Fix a total order on $\mathsf{var}(q^m)$. For every guarded set S in D_q, let $\overline{S}$ be the tuple that contains the variables in S in the fixed order. Now q^s contains, for every maximal guarded set S in D_q, the atom $R_S(\overline{S})$ where R_S is a fresh relation symbol of arity $|S|$. Note that q^s is self-join free, that is, it contains no two different atoms that use the same relation symbol. It is thus a core. Moreover, the Gaifman graph of q^s is identical to that of q^m since the maximal guarded sets of D_{q^m} are exactly those of D_{q^s}. An example of transformation from q to q^s can be found in Figure 3.

We now describe the algorithm. Let a CQ $q^s \in \mathbb{C}$ over schema $\mathbf{S}^s$ and an $\mathbf{S}^s$-database D^s be given as input. To compute $\#q^s(D^s)$, we first enumerate $\mathbb{C}^m$ to find an OMQ $Q = (\mathcal{O}, \mathbf{S}^m, q^m)$ that makes q^s belong to $\mathbb{C}$, as described above. Since Q is a core chased marking, q^m contains $R_x(x)$ for every $x \in \mathsf{var}(q^m)$. Let $\mathbf{S}$ be $\mathbf{S}^m$ without the unary symbols R_x.

(a) CQ q. **(b)** CQ q^s.

Figure 3 A CQ $q() = \exists x \exists y \exists z\ P(x,x,x) \wedge R(x,y) \wedge R(y,z) \wedge R(z,y)$ and it's self-join free counterpart $q^s() = \exists x \exists y \exists z\ R_{xy}(x,y) \wedge R_{yz}(y,z)$ from the proof of Lemma 16.

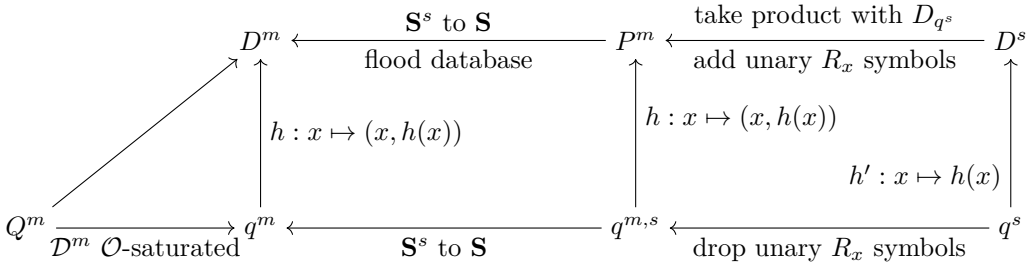


Figure 4 The overall proof strategy of Lemma 16. The diagram depicts the intermediary queries, databases, and underlying relations.

Construct the $\mathbf{S}^s$-database $P = D_{q^s} \times D^s$ and then the $\mathbf{S}^m$-database

$$D^m = \{R(\bar{a}) \mid R \in \mathbf{S} \text{ of arity } |\bar{a}| \text{ and } \bar{a} \text{ tuple over some guarded set } S \text{ in } P\} \cup \{R_x((x,a)) \mid x \in \mathsf{var}(q^m) \text{ and } (x,a) \in \mathsf{dom}(P)\}$$

where a tuple is over set S if it contains only constants from S, in any order and possibly with repetitions. Note that the relations R_x used in the second line are the marking relations from $\mathbf{S}^m$. It is easy to see that the maximal guarded sets of D^m are exactly those of P and that D^m is "flooded" in the sense that we cannot add any facts without introducing a new maximal guarded set. As a consequence and since $\mathcal{O}$ is a set of guarded TGDs, D^m is $\mathcal{O}$*-saturated*, meaning that $p(D^m) = p(\mathsf{ch}_{\mathcal{O}}(D^m))$ for all conjunctive queries p.

Clearly, the database D^m can be constructed within the time requirements of FPT and we can use the oracle to compute $\#Q(D^m)$. Let $q^{s,m}$ be obtained from q^s by adding $R_x(x)$ for every $x \in \mathsf{var}(q^s)$ and let P^m be obtained from P by adding $R_x(x,a)$ for every $a \in \mathsf{adom}(D^s)$. To end the proof, it suffices to show that

$$\#Q(D^m) = \#q^m(D^m) = \#q^{s,m}(P^m) = \#q^s(D^s).$$

The various databases and queries involved as well as the relationships between them are illustrated in Figure 4.

The first equality is immediate since D^m is $\mathcal{O}$-saturated. For the third equality, let $\bar{x} = x_1 \cdots x_n$ be the answer variables in q^s and for any $\bar{a} = a_1 \cdots a_n \in \mathsf{adom}(D^s)^n$, let $\bar{x} \times \bar{a}$ denote the tuple $(x_1,a_1) \cdots (x_n,a_n) \in \mathsf{adom}(P)^n$. Then $q^{s,m}(P^m) = \{\bar{x} \times \bar{a} \mid \bar{a} \in q(D^s)\}$. In fact, this follows from P being the product of D_{q^s} and D^s and from how the relation symbols R_x are used in $q^{s,m}$ and P^m.

It thus remains to deal with the second equality by showing that $q^m(D^m) = q^{s,m}(P^m)$. It is enough to observe that any function $h\colon \mathsf{var}(q^m) \to \mathsf{adom}(D^m)$ is a homomorphism from q^m to D^m if and only if it is a homomorphism from $q^{s,m}$ to P^m.

For the "if" direction, let h be a homomorphism from $q^{s,m}$ to P^m. First let $R(\bar{y})$ be an atom in q^m with $R \in \mathbf{S}$. There is a maximal guarded set S of D_{q^m} that contains all variables in $\bar{y}$. Then $R_S(\overline{S})$ is an atom in q^s and thus $R_S(h(\overline{S})) \in P$. By construction of D^m and since $\bar{y}$ is a tuple over S, this yields $R(h(\bar{y})) \in D^m$, as required. Now let $R_x(x)$ be an atom in q^m. Then $R_x(x)$ is also an atom in $q^{s,m}$ and thus $h(x) \in \{x\} \times \mathsf{adom}(D^s)$ due to the definition of P^m. But then $R_x(h(x)) \in D^m$ by definition of D^m.

For the "only if" direction, let h be a homomorphism from q^m to D^m. First consider atoms $R_S(\overline{S})$ in $q^{s,m}$. Then q^m contains an atom $R(\bar{y})$ where $\bar{y}$ contains exactly the variables in S and thus $R(h(\bar{y})) \in D^m$. By construction of D^m, $h(\bar{y})$ is thus a tuple over some guarded set in P, that is, P contains an atom $Q(\bar{a})$ where $\bar{a}$ contains all constants from $h(\bar{y})$. Let $V \subseteq \mathsf{var}(q^s)$ be the first components of the constants/pairs in $\bar{a}$. Since $Q(\bar{a}) \in P$ and by construction of P, V must be a guarded set in q^s. Now note that we must have $h(y) \in \{y\} \times \mathsf{adom}(D^s)$ for every variable y in $\bar{y}$ due to the use of the relation symbols R_y in q^m and D^m. Thus every variable from $\bar{y}$ (and thus every variable from S) occurs in V and thus $V = S$ because S is a maximal guarded set in D_{q^s}. It follows that $\bar{a}$ uses exactly the constants from $h(\bar{y})$ and not a proper superset. Also, the only atom that uses all variables from S in q^s is $R_S(\overline{S})$ and consequently $Q(\bar{a})$ must be $R_S(h(\overline{S}))$ which is thus in P, as required. It remains to deal with atoms $R_x(x)$ in $q^{s,m}$, which is straightforward as in the "if" direction. ◀

6 Meta Problems

Theorems 7 and 8 show that low values for the structural measures of treewidth, contract treewidth, starsize, and linked matching number are central to efficient answer counting. This suggests the importance of the meta problems to decide whether a given query is equivalent to a query in which some selected structural measures are small, and to construct the latter query if it exists. In the current section, we present some results on this topic both for ontology-mediated querying and for querying under constraints. The obtained results also shed some more light on the interplay between the ontology and the structural measures.

We start with querying under constraints. Our approach is as follows. For a given CQS $(\mathcal{T}, \mathbf{S}, q)$, we construct a certain CQ q' that approximates q from below under the constraints in $\mathcal{T}$ and that has small measures. Then, we show that if there is any CQ q'' that has small measures and is equivalent to q under the constraints in $\mathcal{T}$, then q' is equivalent to q. In this way, we are able to simultaneously solve the decision and computation version of the meta problem at hand. With "approximation from below", we mean that the answers to q' are contained in those to q on all $\mathbf{S}$-databases. This should not be confused with computing a numerical approximation of the number of answers to a given query, which is a very interesting but entirely different problem.

A *set of measures* is a subset $M \subseteq \{\mathrm{TW}, \mathrm{CTW}, \mathrm{SS}, \mathrm{LMN}\}$ with the obvious meaning. For M a set of measures and $k \geq 1$, we say that a UCQ q is an *M_k-query* if for every CQ in q, every measure from M is at most k.

▶ **Definition 17.** *Let $(\mathcal{T}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ be a CQS, M a set of measures, and $k \geq 1$. An* M_k-approximation of q under $\mathcal{T}$ *is a UCQ q' such that*

1. *$q' \subseteq_{\mathcal{T}} q$,*
2. *q' is an M_k-query, and*
3. *for each UCQ q'' that satisfies Conditions 1 and 2, $q'' \subseteq_{\mathcal{T}} q'$.*

We next identify a simple way to construct M_k-approximations. Let $(\mathcal{T}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ be a CQS, M a set of meaures, and $k \geq 1$. Moreover, let ℓ be the maximum number of

variables in any CQ in q. Assuming that $\mathcal{T}$ is understood from the context, we define q_k^M to be the UCQ that contains as a disjunct any CQ p such that $p \subseteq_{\mathcal{T}} q$, p is an M_k-query, and the number of variables in p is bounded by $\ell \cdot \mathsf{ar}(\mathbf{S})$. As containment between UCQs under constraints from $\mathbb{G}$ is decidable [4], given $(\mathcal{T}, \mathbf{S}, q)$ we can effectively compute q_k^M. We observe that q_k^M is an M_k-approximation of q under $\mathcal{T}$.

▶ **Lemma 18.** *Let $(\mathcal{T}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ be a CQS, M a set of measures, and $k \geq 1$. Then q_k^M is an M_k-approximation of q under $\mathcal{T}$.*

Proof. By construction, q_k^M satisfies Points 1 and 2 from Definition 17. We show that it satisfies also Point 3.

We start with some preparations. For an $\mathbf{S}$-database D, a set of TGDs $\mathcal{T}$, and a constant $a \in \mathsf{adom}(\mathsf{ch}_{\mathcal{T}}(D)) \setminus \mathsf{adom}(D)$, a guarded set X over D is a *generator* of a in $\mathcal{T}$ if $a \in \mathsf{adom}(\mathsf{ch}_{\mathcal{T}}(\mathsf{ch}_{\mathcal{T}}(D)|_X))$. Informally, X being a generator of a means that a is located in the tree rooted at X that the chase generates in $\mathsf{ch}_{\mathcal{T}}(D)$. Note that when $\mathcal{T}$ is a set of guarded TGDs, then a generator exists for every $a \in \mathsf{adom}(\mathsf{ch}_{\mathcal{T}}(D)) \setminus \mathsf{adom}(D)$ and furthermore, generators are *complete* in the sense that $R(\bar{a}) \in \mathsf{ch}_{\mathcal{T}}(D)$ with $a \in \bar{a}$ implies $R(\bar{a}) \in \mathsf{ch}_{\mathcal{T}}(\mathsf{ch}_{\mathcal{T}}(D)|_X)$ [12]. We also remark that generators need not be unique.

Let $q''(\bar{x})$ be a UCQ such that $q'' \subseteq_{\mathcal{T}} q$ and q'' is an M_k-query. Further, let p be a CQ in q''. We have to show that q_k^M contains a CQ p' with $p \subseteq_{\mathcal{T}} p'$.

We apply Theorem 10 to the database D_p, the set of TGDs $\mathcal{T}$, and the integer ℓ, defined to be the maximum number of variables of CQs in q. We obtain a database D_p^* which has the properties that $D_p^* \models \mathcal{T}$, $D_p \subseteq D_p^*$, and thus $\bar{x} \in p(D_p^*)$. By containment, $\bar{x} \in q(D_p^*)$, and thus there must be CQ q_i in q such that $\bar{x} \in q_i(D_p^*)$. From Point 2 of Theorem 10 and $|q_i| \leq \ell$, it follows that $\bar{x} \in Q(D_p)$ for the OMQ $Q = (\mathcal{T}, \mathbf{S}, q_i)$. Consequently, q_i maps into $\mathsf{ch}_{\mathcal{T}}(D_p)$ via some homomorphism h that is the identity on $\bar{x}$. We construct a new CQ p' as follows. For each atom $R(\bar{a})$ in q_i,

1. if all constants in $h(\bar{a})$ are from $\mathsf{adom}(p)$, then add $R(h(\bar{a}))$ to p';
2. if $h(\bar{a})$ contains some constant $a \notin \mathsf{adom}(p)$, then take a generator X for a in $\mathcal{T}$ and add all facts in $\mathsf{ch}_{\mathcal{T}}(p)|_X$ as atoms to p'.

The answer variables of p' are exactly those of p. It follows from the construction of p' that the identity is a homomorphism from p' to $\mathsf{ch}_{\mathcal{T}}(p)$. Thus $p \subseteq_{\mathcal{T}} p'$ and it remains to show that p' is a CQ in q_k^M. This follows from the following properties:

1. p' is an M_k-query. Follows from the fact that all guarded sets in p' are also guarded sets in p and thus the Gaifman graph of p' is a subgraph of the Gaifman graph of p, and the fact that all measures are monotone regarding subgraphs.
2. $p' \subseteq_{\mathcal{T}} q_i$. Due to the completeness of generators, the homomorhism h from q_i to $\mathsf{ch}_{\mathcal{T}}(D_p)$ is a also a homomorphism from q_i to $\mathsf{ch}_{\mathcal{T}}(D_p')$. Consequently, $p' \subseteq_{\mathcal{T}} q_i$.
3. $|\mathsf{adom}(p')| \leq \ell \cdot \mathsf{ar}(\mathbf{S})$. For every variable in q_i, at most $\mathsf{ar}(\mathbf{S})$ variables are introduced during the construction of p'. ◀

By definition of M_k-approximations, it is clear that if $(\mathcal{T}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ is a CQS such that q is equivalent under $\mathcal{T}$ to a UCQ q' that is an M_k-query, then any M_k-approximation of q under $\mathcal{T}$ also satisfies these properties. The following is thus an immediate consequence of Lemma 18 and the fact that containment between UCQs under constraints from $\mathbb{G}$ is decidable.

▶ **Theorem 19.** *Let M be a set of measures. Given a CQS $(\mathcal{T}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ and $k \geq 1$, it is decidable whether q is equivalent under $\mathcal{T}$ to a UCQ q' that is an M_k-query. Moreover, if this is the case, then such a q' can be effectively computed.*

A particularly relevant case is $M = \{\mathrm{TW}, \mathrm{CTW}\}$, as it is linked to fixed-parameter tractability. Let $(\mathcal{T}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{CQ})$ and assume that we have computed an equivalent UCQ q' that is an M_k-query as per Theorem 19. Since the original query q is a CQ and $q \equiv_{\mathcal{T}} q'$, there must be a single disjunct q^* of q such that $q \equiv_{\mathcal{T}} q^*$. We can effectively identify q^* and count answers to q^* on any $\mathbf{S}$-database in FPT based on Theorem 4. It remains an interesting open question whether the same is true when the original query is a UCQ and, related to this, whether M_k-approximations always admit answer counting in FPT, that is, even when the original query is not equivalent under $\mathcal{T}$ to an M_k-query.

We now turn to ontology-mediated querying, using essentially the same approach to the meta problems as in the case of CQSs. We say that OMQ $Q_1(\bar{x}) = (\mathcal{O}_1, \mathbf{S}, q_1)$ is *contained* in OMQ $Q_2(\bar{x}) = (\mathcal{O}_2, \mathbf{S}, q_2)$, written $Q_1 \subseteq Q_2$, if $Q_1(D) \subseteq Q_2(D)$ for every $\mathbf{S}$-database D. Q_1 and Q_2 are *equivalent*, written $Q_1 \equiv Q_2$, if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$. We say that an OMQ $Q = (\mathcal{O}, \mathbf{S}, q)$ is an *M_k-query* if q is.

▶ **Definition 20.** *Let $Q = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ be an OMQ, M a set of measures, and $k \geq 1$. An* M_k-approximation of Q *is an OMQ $Q' = (\mathcal{O}', \mathbf{S}, q') \in (\mathbb{G}, \mathbb{UCQ})$ such that*

1. *$Q' \subseteq Q$,*
2. *Q' is an M_k-query, and*
3. *for each $Q'' = (\mathcal{O}'', \mathbf{S}, q'') \in (\mathbb{G}, \mathbb{UCQ})$ that satisfies Conditions 1 and 2, $Q'' \subseteq Q'$.*

We say that Q' is an M_k-approximation of Q while preserving the ontology *if it is an M_k-approximation and $\mathcal{O}' = \mathcal{O}$.*

We show in the appendix of the long version that M_k-approximations of OMQs $(\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ based on the full schema and while preserving the ontology are identical to M_k-approximations of $(\mathcal{O}, \mathbf{S}, q)$ viewed as a CQS. We can thus reuse the approximations from Lemma 18 as a basis for showing the following counterpart of Theorem 19.

▶ **Theorem 21.** *Let M be a set of measures. Given an OMQ $Q = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ based on the full schema and $k \geq 1$, it is decidable whether Q is equivalent to an OMQ $Q' = (\mathcal{O}, \mathbf{S}, q') \in (\mathbb{G}, \mathbb{UCQ})$ that is an M_k-query. Moreover, if this is the case, then such a Q' can be effectively computed.*

We next consider approximations of OMQs that need not preserve the ontology and might not assume the full schema, focussing on single structural measures rather than sets thereof. To simplify notion instead of, say, $\{\mathrm{CTW}\}_k$-approximations, we speak of CTW_k-approximations. We only have full results for contract treewidth and starsize.

A *collapsing* of a CQ $q(\bar{x})$ is a CQ $p(\bar{x})$ that can be obtained from q by identifying variables and adding equality atoms (on answer variables). When an answer variable x is identified with a non-answer variable y, the resulting variable is x; the identification of two answer variables is not allowed. The *CTW_k-approximation* of an OMQ $Q = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$, for $k \geq 1$, is the OMQ $Q_k^{\mathrm{CTW}} = (\mathcal{O}, \mathbf{S}, q_k^{\mathrm{CTW}})$ where q_k^{CTW} is the UCQ that contains as CQs all collapsings of q that have contract treewidth at most k. The *SS_k-approximation* of Q is defined accordingly, and denoted with Q_k^{SS}.

▶ **Theorem 22.** *Let $(\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ be an OMQ and $k \geq 1$. Then Q_k^{CTW} is a CTW_k-approximation of Q. Moreover, if $k \geq \mathsf{ar}(\mathbf{S})$, then Q_k^{SS} is an SS_k-approximation of Q.*

The proof of Theorem 22 is non-trivial and relies on careful manipulations of databases that are tailored towards the structural measure under consideration. It gives rise to decidability results that, in contrast to Theorem 21, neither require the ontology to be preserved nor the schema to be full.

▶ **Corollary 23.** *Given an OMQ $Q = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{UCQ})$ and $k \geq 1$, it is decidable whether Q is equivalent to an OMQ $Q' \in (\mathbb{G}, \mathbb{UCQ})$ of contract treewidth at most k. Moreover, if this is the case, then such a Q' can be effectively computed. The same is true for starsize in place of contract treewidth.*

For treewidth, we leave open decidability of the meta problem and only observe that it behaves differently from contract treewidth and starsize in that obtaining approximations might require a modification of the ontology. This is even true when the schema is full.

▶ **Example 24.** For $n \geq 3$, let $Q_n() = (\emptyset, \mathbf{S}_n, q_n \vee p_n)$ where $\mathbf{S}_n = \{W, R_1, \ldots, R_n\}$ with W of arity n and each R_i binary and where

$$q_n = \exists x_1 \cdots \exists x_n \, W(x_1, \ldots, x_n) \quad \text{and} \quad p_n = \exists x_1 \cdots \exists x_n \exists y \, R_1(x_1, y), \ldots, R_n(x_n, y).$$

Then $Q'_n() = (\mathcal{O}, \mathbf{S}_n, p_n)$ with $\mathcal{O} = \{W(\bar{x}) \to p_n(\bar{x})\}$ is a TW_1-approximation of Q_n. In fact, it is equivalent to Q_n. However, Q_n has no TW_k-approximation Q^* based on the same (empty) ontology as Q_n for any $k < n$ since $Q'_n \not\subseteq Q^*$ for any $Q^* = (\emptyset, \mathbf{S}_n, q^*)$ such that $Q^* \subseteq Q$ and $q*$ is of treewidth $k < n$. In fact, any Q^* with the latter property does not return any answers on the database $\{W(a_1, \ldots, a_n)\}$.

In the appendix of the long version, we provide a further set of examples which does not require the arity of relation names to grow. It does, however, use a data schema that is not full. It remains an interesting and non-trival open problem to prove a counterpart of Corollary 23, even for the case of the full schema.

7 Conclusions

We have provided a complexity classification for counting the number of answers to UCQs in the presence of TGDs that applies both to ontology-mediated querying and to querying under constraints. The classification also applies to ontology-mediated querying with the OMQ language $(\mathcal{ELIH}, \mathrm{UCQ})$ where $\mathcal{ELIH}$ is a well-known description logic [2]. In fact, this is immediate if the ontologies in OMQs are in a certain well-known normal form that avoids nesting of concepts [2]. In the general case, it suffices to observe that all our proofs extended from guarded TGDs to frontier-guarded TGDs [3] with bodies of bounded treewidth, a strict generalization of $\mathcal{ELIH}$. In contrast, a complexity classification for OMQs based on frontier-guarded TGDs with unrestricted bodies is an interesting problem for future work.

There are several other interesting questions that remain open, we mention only a few. In querying under constraints that are guarded TGDs, does answer counting in FPT coincide with answer counting in PTime? Do our results extend to ontology-mediated querying when the data schema is not required to be full? What about OMQs and CQSs based on other decidable classes of TGDs? And how can we decide the meta problems for the important structural measure of treewidth when the ontology needs not be preserved, with full data schema or even with unrestricted data schema?

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

2 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017. `doi:10.1017/9781139025355`.

3 Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011. doi:10.1016/j.artint.2011.03.002.

4 Pablo Barceló, Gerald Berger, and Andreas Pieris. Containment for rule-based ontology-mediated queries. In *PODS*, pages 267–279, 2018. doi:10.1145/3196959.3196963.

5 Pablo Barceló, Victor Dalmau, Cristina Feier, Carsten Lutz, and Andreas Pieris. The limits of efficiency for open- and closed-world query evaluation under guarded TGDs. In *Proc. of PODS*, 2020. doi:10.1145/3375395.3387653.

6 Pablo Barceló, Cristina Feier, Carsten Lutz, and Andreas Pieris. When is ontology-mediated querying efficient? In *Proc. of LICS*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785823.

7 Pablo Barceló, Diego Figueira, Georg Gottlob, and Andreas Pieris. Semantic optimization of conjunctive queries. *J. ACM*, 67(6), 2020. doi:10.1145/3424908.

8 Pablo Barceló, Georg Gottlob, and Andreas Pieris. Semantic acyclicity under constraints. In *PODS*, pages 343–354, 2016. doi:10.1145/2902251.2902302.

9 Meghyn Bienvenu, Quentin Manière, and Michaël Thomazo. Answering counting queries over DL-Lite ontologies. In *Proc. of IJCAI*, 2020. doi:10.24963/ijcai.2020/223.

10 Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Proc. of Reasoning Web*, volume 9203 of *LNCS*, pages 218–307. Springer, 2015. doi:10.1007/978-3-319-21768-0_9.

11 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014. doi:10.1145/2661643.

12 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013. doi:10.1613/jair.3873.

13 Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012. doi:10.1016/j.artint.2012.08.002.

14 Diego Calvanese, Julien Corman, Davide Lanti, and Simon Razniewski. Counting query answers over DL-Lite knowledge base. In *Proc. of IJCAI*, 2020. doi:10.24963/ijcai.2020/230.

15 Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS*, pages 149–158. ACM Press, 1998. doi:10.1145/275487.275504.

16 Hubie Chen and Stefan Mengel. A trichotomy in the complexity of counting answers to conjunctive queries. In *Proc. of ICDT*, pages 110–126, 2015. doi:10.4230/LIPIcs.ICDT.2015.110.

17 Hubie Chen and Stefan Mengel. Counting answers to existential positive queries: A complexity classification. In *Proc. of PODS*, pages 315–326, 2016. doi:10.1145/2902251.2902279.

18 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *J. Theor. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

19 Holger Dell, Marc Roth, and Philip Wellnitz. Counting answers to existential questions. In *Proc. of ICALP*, pages 113:1–113:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.113.

20 Arnaud Durand and Stefan Mengel. The complexity of weighted counting for acyclic conjunctive queries. *J. Comput. Syst. Sci.*, 80(1):277–296, 2014. doi:10.1016/j.jcss.2013.08.001.

21 Arnaud Durand and Stefan Mengel. Structural tractability of counting of solutions to conjunctive queries. *J. Theory Comput. Syst.*, 57(4):1202–1249, 2015. doi:10.1007/s00224-014-9543-y.

22 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *J. Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.

23 Cristina Feier, Carsten Lutz, and Marcin Przybyłko. Answer counting under guarded TGDs, 2021. arXiv:2101.03058.

24 Diego Figueira. Semantically acyclic conjunctive queries under functional dependencies. In *Proc. of LICS*, page 847–856. ACM, 2016. doi:10.1145/2933575.2933580.

25 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

26 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

27 David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984. doi:10.1016/0022-0000(84)90081-3.

28 Bogdan Kostov and Petr Kremen. Count distinct semantic queries over multiple linked datasets. *OJSW*, 5(1):1–11, 2018. URL: http://nbn-resolving.de/urn:nbn:de:101:1-201712245426.

29 Egor V. Kostylev and Juan L. Reutter. Complexity of answering counting aggregate queries over DL-Lite. *J. Web Semant.*, 33:94–111, 2015. doi:10.1016/j.websem.2015.05.003.

30 Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. Fast query answering over existential rules. *ACM Trans. Comput. Log.*, 20(2):12:1–12:48, 2019. doi:10.1145/3308448.

31 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979. doi:10.1145/320107.320115.

32 Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *J. Comput. Syst. Sci.*, 79(6):984–1001, 2013. doi:10.1016/j.jcss.2013.01.012.

33 Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 4900:133–173, 2008. doi:10.1007/978-3-540-77688-8_5.

Maximum Coverage in the Data Stream Model: Parameterized and Generalized

Andrew McGregor
University of Massachusetts Amherst, MA, USA

David Tench
Stony Brook University, NY, USA

Hoa T. Vu
San Diego State University, CA, USA

Abstract

We present algorithms for the `Max Coverage` and `Max Unique Coverage` problems in the data stream model. The input to both problems are m subsets of a universe of size n and a value $k \in [m]$. In `Max Coverage`, the problem is to find a collection of at most k sets such that the number of elements covered by at least one set is maximized. In `Max Unique Coverage`, the problem is to find a collection of at most k sets such that the number of elements covered by exactly one set is maximized. These problems are closely related to a range of graph problems including matching, partial vertex cover, and capacitated maximum cut. In the data stream model, we assume k is given and the sets are revealed online. Our goal is to design single-pass algorithms that use space that is sublinear in the input size. Our main algorithmic results are:

- If the sets have size at most d, there exist single-pass algorithms using $O(d^{d+1}k^d)$ space that solve both problems exactly. This is optimal up to polylogarithmic factors for constant d.
- If each element appears in at most r sets, we present single pass algorithms using $\tilde{O}(k^2r/\epsilon^3)$ space that return a $1+\epsilon$ approximation in the case of `Max Coverage`. We also present a single-pass algorithm using slightly more memory, i.e., $\tilde{O}(k^3r/\epsilon^4)$ space, that $1+\epsilon$ approximates `Max Unique Coverage`.

In contrast to the above results, when d and r are arbitrary, any constant pass $1+\epsilon$ approximation algorithm for either problem requires $\Omega(\epsilon^{-2}m)$ space but a single pass $O(\epsilon^{-2}mk)$ space algorithm exists. In fact any constant-pass algorithm with an approximation better than $e/(e-1)$ and $e^{1-1/k}$ for `Max Coverage` and `Max Unique Coverage` respectively requires $\Omega(m/k^2)$ space when d and r are unrestricted. En route, we also obtain an algorithm for a parameterized version of the streaming `Set Cover` problem.

2012 ACM Subject Classification Theory of computation → Sketching and sampling; Theory of computation → Approximation algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Data streams, maximum coverage, maximum unique coverage, set cover

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.12

Funding This work was partially supported by NSF grants CCF-1934846, CCF-1908849, and CCF-1637536.

1 Introduction

Problem Description. We consider the `Max Coverage` and `Max Unique Coverage` problems in the data stream model. The input to both problems are m subsets of a universe of size n and a value $k \in [m]$. In `Max Coverage`, the problem is to find a collection of at most k sets such that the number of elements covered by at least one set is maximized. In `Max Unique Coverage`, the problem is to find a collection of at most k sets such that the

number of elements covered by exactly one set is maximized. In the data stream model, we assume k is provided but that the sets are revealed online and our goal is to design single-pass algorithms that use space that is sub-linear in the input size.

`Max Coverage` is a classic `NP-Hard` problem that has a wide range of applications including facility and sensor allocation [51], information retrieval [5], influence maximization in marketing strategy design [47], and the blog monitoring problem [63]. It is well-known that the greedy algorithm, which greedily picks the set that covers the most number of uncovered elements, is a $e/(e-1)$ approximation and that unless `P` = `NP`, this approximation factor is the best possible in polynomial time [30].

`Max Unique Coverage` was first studied in the offline setting by Demaine et al. [25]. A motivating application for this problem was in the design of wireless networks where we want to place base stations that cover mobile clients. Each station could cover multiple clients but unless a client is covered by a unique station the client would experience too much interference. Demaine et al. [25] gave a polynomial time $O(\log k)$ approximation. Furthermore, they showed that `Max Unique Coverage` is hard to approximate within a factor $O(\log^{\sigma} n)$ for some constant σ under reasonable complexity assumptions. Erlebach and van Leeuwen [29] and Ito et al. [40] considered a geometric variant of the problem and Misra et al. [61] considered the parameterized complexity of the problem. This problem is also closely related to Minimum Membership Set Cover where one has to cover every element and minimizes the maximum overlap on any element [26, 52].

In the streaming set model, `Max Coverage` and the related `Set Cover` problem[1] have both received a significant amount of attention [7, 15, 27, 36, 38, 39, 60, 63]. The most relevant result is a single-pass $2+\epsilon$ approximation using $\tilde{O}(k\epsilon^{-3})$ space [8,60] although better approximation is possible in a similar amount of space if multiple passes are permitted [60] or if the stream is randomly ordered [2, 62]. In this paper, we almost exclusively consider single-pass algorithms where the sets arrive in an arbitrary order.

The unique coverage problem has not been studied in the data stream model although it, and `Max Coverage`, are closely related to various graph problems that have been studied.

Relationship to Graph Streaming. There are two main variants of the graph stream model. In the *arbitrary order model*, the stream consists of the edges of the graph in arbitrary order. In the *adjacency list model*, all edges that include the same node are grouped together. Both models generalize naturally to hypergraphs where each edge could consists of more than two nodes. The arbitary order model has been more heavily studied than the adjacency list model but there has still been a significant amount of work in the latter model [6, 7, 11, 36, 41, 49, 56–58]. For further details, see a recent survey on work on the graph stream model [55].

To explore the relationship between `Max Coverage` and `Max Unique Coverage` and various graph stream problems, it makes sense to introduce to additional parameters beyond m (the number of sets) and n (the size of the universe). Specifically, throughout the paper we let d denote the maximum cardinality of a set in the input and let r denote the maximum multiplicity of an element in the universe where the *multiplicity* is the number of sets an element appears.[2] Then an input to `Max Coverage` and `Max Unique Coverage` can define a (hyper)graph in one of the following two natural ways:

[1] That is, find the minimum number of sets that cover the entire universe.

[2] Note that d and r are dual parameters in the sense that if the input is $\{S_1, \ldots, S_m\}$ and we define $T_i = \{j : i \in S_j\}$ then $d = \max_j |S_j|$ and $r = \max_i |T_i|$.

1. *First Interpretation:* A sequence of (hyper-)edges on a graph with n nodes of maximum degree r (where the degree of a node v corresponds to how many hyperedges include that node) and m hyperedges where each hyperedge has size at most d. In the case where every set has size $d = 2$, the hypergraph is an *ordinary graph*, i.e., a graph where every edge just has two endpoints. With this interpretation, the graph is being presented in the arbitrary order model.
2. *Second Interpretation:* A sequence of adjacency lists (where the adjacency list for a given node includes all the hyperedges that include that node) on a graph with m nodes of maximum degree d and n hyperedges of maximum size r. In this interpretation, if every element appears in exactly $r = 2$ sets, then this corresponds to an ordinary graph where each element corresponds to an edge and each set corresponds to a node. With this interpretation, the graph is being presented in the adjacency list model.

Under the first interpretation, the `Max Coverage` problem and the `Max Unique Coverage` problem when all sets have size exactly 2 naturally generalize the problem of finding a maximum matching in an ordinary graph in the sense that if there exists a matching with at least k edges, the optimum solution to either `Max Coverage` and `Max Unique Coverage` will be a matching. There is a large body of work on graph matchings in the data stream model [3, 12, 23, 24, 28, 31, 34, 35, 42, 43, 48–50, 54, 65] including work specifically on solving the problem exactly if the matching size is bounded [18, 20]. More precisely, `Max Coverage` corresponds to the partial vertex cover problem [53]: what is the maximum number of edges that can be covered by selecting k nodes. For larger sets, the `Max Coverage` and `Max Unique Coverage` are at least as hard as finding partial vertex covers and matching in hypergraphs.

Under the second interpretation, when all elements have multiplicity 2, then the problem `Max Unique Coverage` corresponds to finding the capacitated maximum cut, i.e., a set of at most k vertices such that the number of edges with exactly one endpoint in this set is maximized. In the offline setting, Ageev and Sviridenko [1] and Gaur et al. [33] presented a 2 approximation for this problem using linear programming and local search respectively. The (uncapacitated) maximum cut problem was been studied in the data stream model by Kapralov et al. [44–46]; a 2-approximation is trivial in logarithmic space[3] but improving on this requires space that is polynomial in the size of the graph. The capacitated problem is a special case of the problem of maximizing a non-monotone sub-modular function subject to a cardinality constraint. This general problem has been considered in the data stream model [8, 13, 16, 37] but in that line of work it is assumed that there is oracle access to the function being optimized, e.g., given any set of nodes, the oracle will return the number of edges cut. Alaluf et al. [4] presented a $2+\epsilon$ approximation in this setting, assuming exponential post-processing time. In contrast, our algorithm does not assume an oracle while obtaining a $1 + \epsilon$ approximation (and also works for the more general problem `Max Unique Coverage`).

1.1 Our Results

Our main results are the following single-pass streaming algorithms[4]:

(A) Bounded Set Cardinality. If all sets have size at most d, there exists a $\tilde{O}(d^{d+1}k^d)$ space data stream algorithm that solves `Max Unique Coverage` and `Max Coverage` exactly. We show that this is nearly optimal in the sense that any exact algorithm requires $\Omega(k^d)$ space for constant d.

[3] It suffices to count the number of edges M since there is always a cut whose size is at least $M/2$.
[4] Throughout we use $\tilde{O}$ to denote that logarithmic factors of m and n are being omitted.

(B) Bounded Multiplicity. If every element appears in at most r sets, we present the following algorithms:

- (B1) `Max Unique Coverage`: There exists a $1+\epsilon$ approximation using $\tilde{O}(\epsilon^{-4}k^3r)$ space.
- (B2) `Max Coverage`: There exists a $1+\epsilon$ approximation algorithm using $\tilde{O}(\epsilon^{-3}k^2r)$ space.

In contrast to the above results, when d and r are arbitrary, any constant pass $1+\epsilon$ approximation algorithm for either problem requires $\Omega(\epsilon^{-2}m)$ space [6].[5] We also generalize of lower bound for `Max Coverage` [60] to `Max Unique Coverage` to show that any constant-pass algorithm with an approximation better than $e^{1-1/k}$ requires $\Omega(m/k^2)$ space. We also present a single-pass algorithm with an $O(\log \min(k, r))$ approximation for `Max Unique Coverage` using $\tilde{O}(k^2)$ space, i.e., the space is independent of r and d but the approximation factor depends on r. This algorithm is a simple combination of a `Max Coverage` algorithm due to McGregor and Vu [60] and an algorithm for `Max Unique Coverage` in the offline setting due to Demaine et al. [25]. Finally, our `Max Coverage` result (B2) algorithm also yields a new multi-pass result for a parameterized version of the streaming `Set Cover` problem. We will also show that results (A) and (B2) can also be made to handle stream deletions. The generalization for result (A) that we present requires space that scales with k^{2d} rather than k^d. However, in subsequent work we have shown that space the scales with k^d is also sufficient in the insert/delete setting.

1.2 Technical Summary and Comparisons

Technical Summary. Our results are essentially streamable kernelization results, i.e., the algorithm "prunes" the input (in the case of `Max Unique Coverage` and `Max Coverage` this corresponds to ignoring some of the input sets) to produce a "kernel" in such a way that a) solving the problem optimally on the kernel yields a solution that is as good (or almost as good) as the optimal solution on the original input and b) the kernel can be constructed in the data stream model and is sufficiently smaller than the original input such that it is possible to find an optimal solution for the kernel in significantly less time than it would take to solve on the original input. In the field of fixed parameter tractability, the main requirement is that the kernel can be produced in polynomial time. In the growing body of work on streaming kernelization [17–19] the main requirement is that the kernel can be constructed using small space in the data stream model. Our results fits in with this line of work and the analysis requires numerous combinatorial insights into the structure of the optimum solution for `Max Unique Coverage` and `Max Coverage`.

Our technical contributions can be outlined as follows.

- Result (A) relies on a key combinatorial lemma. This lemma provides a rule to discard sets such that there is an optimum solution that does not contain any of the discarded sets. Furthermore, the number of stored sets can be bounded in terms of k and d.
- Result (B1) uses the observation that each set of any optimal solution intersects some maximal collection of disjoint sets. The main technical step is to demonstrate that storing a small number of intersecting sets, in terms of k and r, suffices to preserve the optimal solution.

[5] The lower bound result by Assadi [6] was for the case of `Max Coverage` but we will explain that it also applies in the case of `Max Unique Coverage`.

- Result (B2) is based on a very simple idea of first collecting the largest $O(rk/\epsilon)$ sets and then solving the problem optimally on these sets. This can be done in a space efficient manner using existing sketch for F_0 estimation in the case of `Max Coverage`. While the approach is simple, showing that it yields the required approximations requires some work that builds on a recent result by Manurangsi [53]. We also extend the algorithm to the model where sets can be inserted and deleted.

Comparison to Related Work. In the context of streaming algorithms, for the `Max Coverage` problem, McGregor and Vu [59] showed that any approximation better than $e/(e-1)$ requires $\Omega(m/k^2)$ space. For the more general problem of streaming submodular maximization subject to a cardinality constraint, Feldman et al. [32] very recently showed a stronger lower bound that any approximation better than 2 requires $\Omega(m)$ space. Our results provide a route to circumvent these bounds via parameterization on k, r, and d.

Result (B2) also leads to a parameterized algorithm for streaming `Set Cover`. This new algorithm uses $\tilde{O}(rk^2n^\delta + n)$ space which improves upon the algorithm by Har-Peled et al. [36] that uses $\tilde{O}(mn^\delta + n)$ space, where k is an upper bound for the size of the minimum set cover, in the case $rk^2 \ll m$. Both algorithms use $O(1/\delta)$ passes and yield an $O(1/\delta)$ approximation.

In the context of offline parameterized algorithms, Bonnet et al. [10] showed that `Max Coverage` is fixed-parameter tractable in terms of k and d. However, their branching-search algorithm cannot be implemented in the streaming setting. Misra et al. [61] showed that the maximum unique coverage problem in which the aim is to maximize the number of uniquely covered elements u (without any restriction on the number of sets) admits a kernel of size 4^u. On the other hand, they showed that the budgeted version of this problem (where each element has a profit and each set has a cost and the goal is maximize the profit subject to a budget constraint) is $W[1]$-hard when parameterized by the budget[6]. In this context, our result shows that a parameterization on both the maximum set size d and the budget k is possible (at least when all costs and profits are unit).

2 Preliminaries

2.1 Notation and Parameters

Throughout the paper, m will denote the number of sets, n will denote the size of the universe, and k will denote the maximum number of sets that can be used in the solution. Given input sets $S_1, S_2, \ldots, S_m \subseteq [n]$, let

$$d = \max_i |S_i|$$

be the maximum set size and let

$$r = \max_j |\{i : j \in S_i\}|$$

be the maximum number of sets that contain the same element.

Suppose C is a collection of sets. We let $F(C)$ (and $G(C)$) be the set of elements covered (and uniquely covered) by an optimal solution in C. Furthermore, let $f(C) = |F(C)|$ and $g(C) = |G(C)|$. In other words, $f(C)$ is the maximum number of elements that can be

[6] In the `Max Unique Coverage` problem that we consider, all costs and profits are one and the budget is k.

covered by k sets. Similarly, $g(C)$ is the maximum number of elements that can be uniquely covered by k sets. Furthermore, let $\psi(C)$ and $\tilde{\psi}(C)$ be the set of elements covered and uniquely covered respectively by the sets in C.

To ease the notation, if C is a collection of set and S is a set, we often use $C - S$ to denote $C \setminus \{S\}$ and $C + S$ to denote $C \cup \{S\}$.

We use M to denote the collection of all sets in the stream. Therefore, the optimal value to `Max Coverage` and `Max Unique Coverage` are $f(M)$ and $g(M)$ respectively.

Throughout this paper, we say an algorithm is correct with high probability if the probability of failure is inversely polynomial in m.

2.2 Sketches and Subsampling

Coverage Sketch. Given a vector $x \in \mathbb{R}^n$, $F_0(x)$ is defined as the number of elements of x which are non-zero. If given a subset $S \subset \{1, \ldots, n\}$, we define $x_S \in \{0,1\}^n$ to be the characteristic vector of S (i.e., $x_i = 1$ iff $i \in S$) then given sets $S_1, S_2, \ldots$ note that $F_0(x_{S_1} + x_{S_2} + \ldots)$ is exactly the number of elements covered by $S_1 \cup S_2 \cup \ldots$. We will use the following result for estimating F_0.

▶ **Theorem 1** (F_0 Sketch [9, 21]). *Given a set $S \subseteq [n]$, there exists an $\tilde{O}(\epsilon^{-2} \log \delta^{-1})$-space algorithm that constructs a data structure $\mathcal{M}(S)$ (called an F_0 sketch of S). The sketch has the property that the number of distinct elements in a collection of sets $S_1, S_2, \ldots, S_t$ can be approximated up to a $1+\epsilon$ factor with probability at least $1-\delta$ provided the collection of F_0 sketches $\mathcal{M}(S_1), \mathcal{M}(S_2), \ldots, \mathcal{M}(S_t)$.*

Note that if we set $\delta \ll 1/(\text{poly}(m) \cdot \binom{t}{k})$ in the above result we can try each collection of k sets amongst $S_1, S_2, \ldots, S_t$ and get a $1+\epsilon$ approximation for the coverage of each collection with high probability.

Unique Coverage Sketch. For unique coverage, our sketch of a set corresponds to subsampling the universe via some hash function $h : [n] \to \{0,1\}$ where h is chosen randomly such that for each i, $\Pr[h(i) = 1] = p$ for some appropriate value p. Specifically, rather processing an input set S, we process $S' = \{i \in S : h(i) = 1\}$. Note that $|S'|$ has size $p|S|$ in expectation. This approach was use by McGregor and Vu [60] in the context of `Max Coverage` and it extends easily to `Max Unique Coverage`; see Section 7. The consequence is that if there is a streaming algorithm that finds a t approximation, we can turn that algorithm into a $t(1+\epsilon)$ approximation algorithm in which we can assume that $\text{OPT} = O(\epsilon^{-2} k \log m)$ with high probability by running the algorithm on a subsampled sets rather than the original sets. Note that this also allows us to assume input sets have size $O(\epsilon^{-2} k \log m)$ since $|S'| \leq \text{OPT}$. Hence each "sketches" set can be stored using $B = O(\epsilon^{-2} k \log m \log n)$ bits.

An Algorithm with $\tilde{O}(\epsilon^{-2}mk)$ Memory. We will use the above sketches in a more interesting context later in the paper, but note that they immediately imply a trivial algorithmic result. Consider the naive algorithm that stores every set and finds the best solution; note that this requires exponential time. We note that since we can assume $\text{OPT} = O(\epsilon^{-2} k \log m)$, each set has size at most $O(\epsilon^{-2} k \log m)$. Hence, we need $\tilde{O}(\epsilon^{-2} mk)$ memory to store all the sets. This approach was noted in [60] in the context of `Max Coverage` but also apples to `Max Unique Coverage`. We will later show that for a $1+\epsilon$ approximation, the above trivial algorithm is optimal up to polylogarithmic factors for constant k.

3 An Exact Algorithm

Algorithm. Our algorithm, though perhaps non-intuitive, is simple to state:

1. Initialize X to be an empty collection of sets. Let $b = d(k-1)$.
2. Let X_a be the sub-collection of X that contains sets of size a.
3. For each set S in the stream: Suppose $|S| = a$. Add S to X if there does not exist $T \subseteq S$ that occurs as a subset of $(b+1)^{d-|T|}$ sets of X_a.
4. Post-processing: Return the best solution C in X.

Analysis. Our algorithm relies on the following combinatorial lemma.

▶ **Lemma 2.** *Let $W = \{S_1, S_2, \ldots\}$ be a collection of distinct sets where each $S_i \subseteq [n]$ and $|S_i| = a$. Suppose for all $T \subseteq \psi(W)$ with $|T| \leq a$ there exist at most*

$$\ell_{|T|} := (b+1)^{a-|T|}$$

sets in W that contain T. Furthermore, suppose there exists a set T^* such that this inequality is tight. Then, for all $B \subseteq \psi(W)$ disjoint from T^* with $|B| \leq b$ there exists a set $Y \in W$ such that $T^* \subseteq Y$ and $|Y \cap B| = 0$.

Proof. If $|T^*| = a$ then $T^* \in W$, then we can simply set $Y = T^*$. Henceforth, assume $|T^*| < a$. Consider the $\ell_{|T^*|}$ sets in W that are supersets of T^*. Call this collection W'. For any $x \in B$, there are at most $\ell_{|T^*|+1}$ sets that include $T^* \cup \{x\}$. Since there are b choices for x, at most

$$b\ell_{|T^*|+1} = b(b+1)^{a-|T^*|-1} < (b+1)^{a-|T^*|} = \ell_{|T^*|}$$

sets in W' contain an element in B. Hence, at least one set Y in W' does not contain any element in B. ◀

We show that the algorithm indeed obtains an exact kernel for the problems. Recall that M is the collection of all sets in the stream, i.e., the optimal solution has size $f(M)$.

▶ **Theorem 3.** *The output of the algorithm is optimal. In particular, $f(C) = f(M)$ and $g(C) = g(M)$.*

Proof. Recall that X is the collection of all stored sets. We define

$$C_i = M \setminus \{\text{the first } i \text{ sets in the stream that are not stored in } X\}.$$

Clearly, $f(C_0) = f(M)$. Now, suppose there exists $i \geq 1$ such that $f(C_i) < f(M)$. Let i be the smallest such index. Let $\mathcal{O}$ be an optimal solution of C_{i-1} (note that $\mathcal{O}$ is also an overall optimal solution based on the minimal assumption on i). Let S be the ith set that was not stored in X. If $S \notin \mathcal{O}$ then we have a contradiction since $f(C_i) = f(C_{i-1}) = f(M)$. Thus, assume $S \in \mathcal{O}$. Suppose $|S| = a$.

▷ Claim 4. There exists Y in X_a such that $f(\mathcal{O} - S + Y) \geq f(\mathcal{O})$.

Proof. Note that S was not stored because there existed $T^* \subseteq S$ such that T^* was a subset of $(b+1)^{d-|T^*|}$ sets in X_a. Consider the set $B = \psi(\mathcal{O}) \setminus S$. Clearly, $B \cap T^* = \emptyset$ and $|B| \leq d(k-1)$. By Lemma 2, there is a set Y in X_a such that $Y \cap B = \emptyset$.

Let $Y' = Y \setminus S$ and $S' = S \setminus Y$. Note that $|Y'| = |S'|$ since $|Y| = |S|$. Define indicator variables $\alpha_z = 1$ iff $z \in \psi(\mathcal{O} - S + Y)$ and $\beta_z = 1$ iff $z \in \psi(\mathcal{O})$. Note that

$$\begin{aligned}(z \in Y \cap S \text{ or } z \notin Y \cup S) &\implies (\alpha_z = \beta_z)\,,\\ (z \in Y') &\implies (\alpha_z = 1)\,,\\ (z \in Y') &\implies (\beta_z = 0)\,,\end{aligned}$$

where the last equation uses the fact that Y' is disjoint from $\psi(\mathcal{O})$. Then

$$\begin{aligned}|\psi(\mathcal{O} - S + Y)| &= \sum_{z \in Y'} \alpha_z + \sum_{z \in Y \cap S} \alpha_z + \sum_{z \in S'} \alpha_z + \sum_{z \notin Y \cup S} \alpha_z\\ &\geq \left(|Y'| + \sum_{z \in Y'} \beta_z\right) + \sum_{z \in Y \cap S} \beta_z + \left(-|S'| + \sum_{z \in S'} \beta_z\right) + \sum_{z \notin Y \cup S} \beta_z\\ &= \sum_{z \in Y'} \beta_z + \sum_{z \in Y \cap S} \beta_z + \sum_{z \in S'} \beta_z + \sum_{z \notin Y \cup S} \beta_z = |\psi(\mathcal{O})|\,.\end{aligned}$$

◁

Thus, $f(C_i) \geq f(\mathcal{O}) = f(M)$ which is a contradiction. Hence, there is no such i and the claim follows. The proof for unique coverage is almost identical: for the analogous claim we define indicator variables $\tilde{\alpha}_z = 1$ iff $z \in \tilde{\psi}(\mathcal{O} - S + Y)$ and $\tilde{\beta}_z = 1$ iff $z \in \tilde{\psi}(\mathcal{O})$. The proof goes through with α and β replaced by $\tilde{\alpha}$ and $\tilde{\beta}$ since it is still the case that

$$\begin{aligned}(z \in Y \cap S \text{ or } z \notin Y \cup S) &\implies \left(\tilde{\alpha}_z = \tilde{\beta}_z\right)\,,\\ (z \in Y') &\implies (\tilde{\alpha}_z = 1)\,,\\ (z \in Y') &\implies \left(\tilde{\beta}_z = 0\right)\,,\end{aligned}$$

where now the last two equations use the fact that Y' is disjoint from $\psi(\mathcal{O})$. ◀

▶ **Lemma 5.** *The space used by the algorithm is $\tilde{O}(d^{d+1}k^d)$.*

Proof. Recall that one of the requirements for a set S to be added to X is that the number of sets in $X_{|S|}$ that are supersets of any subset of S of size t is at most $(b+1)^{d-t}$. This includes the empty subset and since every set in $X_{|S|}$ is a superset of the empty set, we deduce that $|X_{|S|}| \leq (b+1)^d = O((dk)^d)$. Since each set needs $\tilde{O}(d)$ bits to store, and $|X| = \sum_{a=1}^{d} |X_a| \leq O(d^d k^d)$, the total space is $\tilde{O}(d^{d+1}k^d)$. ◀

We summarize the above as a theorem.

▶ **Theorem 6.** *There exist single-pass algorithms using $\tilde{O}(k^d d^{d+1})$ space that yields an exact solution to* `Max Coverage` *and* `Max Unique Coverage`*.*

Handling Insertion-Deletion Streams. We outline another exact algorithm that works for insertion-deletion streams, however with a worse space bound $\tilde{O}((kd)^{2d})$, in Section 6.1.

▶ **Theorem 7.** *There exist randomized single-pass algorithms using $\tilde{O}((kd)^{2d})$ space and allowing deletions that w.h.p. yield an exact solution to* `Max Coverage` *and* `Max Unique Coverage`*.*

4 Approximation Algorithms

In this section, we present a variety of different approximation algorithms where the space used by the algorithm is independent of d but, in some cases, may depend on r. The first algorithm uses $\tilde{O}(\epsilon^{-4}k^3 r)$ memory and obtains a $1+\epsilon$ approximation to both problems. The second algorithm uses $\tilde{O}(\epsilon^{-3}k^2 r)$ memory and obtains a $1+\epsilon$ approximation to `Max Coverage` and a $2+\epsilon$ approximation to `Max Unique Coverage`; it can also be extended to streams with deletions.

4.1 A $1+\epsilon$ Approximation

Given a collection of sets $C = \{S_1, S_2, \ldots, S_m\}$, we say a sub-collection $C' \subset C$ is a *matching* if the sets in C' are mutually disjoint. C' is a maximal matching if there does not exist $S \in C \setminus C'$ such that S is disjoint from all sets in C'.

▶ **Lemma 8.** *For any input C, let $O \subset C$ be an optimal solution for either the* `Max Coverage` *or* `Max Unique Coverage` *problem. Let M_i be a maximal matching amongst the input set of size i. Then every set of size i in O intersects with some set in M_i.*

Proof. Let $S \in O$ have size i. If it was disjoint from all sets in M_i then it could be added to M_i and the resulting collection would still be a matching. This violates the assumption that M_i is maximal. ◀

The next lemma extends the above result to show that we can potentially remove many sets from each M_i and still argue that there is an optimal solution for the original instance amongst the sets that intersect a set in some M_i.

▶ **Lemma 9.** *Consider an input of sets of size at most d. For $i \in [d]$, let M_i be a maximal matching amongst the input set of size i and let M'_i be an arbitrary subset of M_i of size $\min(k+dk, |M_i|)$. Let D_i be the collection of all sets that intersect a set in M'_i. Then $\bigcup_i (D_i \cup M'_i)$ contains an optimal solution to both the* `Max Unique Coverage` *and* `Max Coverage` *problem.*

Proof. If $|M_i| = |M'_i|$ for all $1 \le i \le d$ then the result follows from Lemma 8. If not, let $j = \max\{i \in [d] : |M_i| > |M'_i|\}$. Let $\mathcal{O}$ be an optimal solution and let $\mathcal{O}_i$ be all the sets in $\mathcal{O}$ of size i. We know that every set in $\mathcal{O}_d \cup \mathcal{O}_{d-1} \cup \ldots \cup \mathcal{O}_{j+1}$ is in

$$\bigcup_{i \ge j+1} (D_i \cup M'_i) = \bigcup_{i \ge j+1} (D_i \cup M_i) \ .$$

Hence, the number of elements (uniquely) covered by $\mathcal{O}$ is at most the number of elements (uniquely) covered by $\mathcal{O}_d \cup \mathcal{O}_{d-1} \cup \ldots \cup \mathcal{O}_{j+1}$ plus kj since every set in $\mathcal{O}_j \cup \ldots \cup \mathcal{O}_1$ (uniquely) covers at most j additional elements. But we can (uniquely) cover at least the number of elements (uniquely) covered by $\mathcal{O}_d \cup \mathcal{O}_{d-1} \cup \ldots \cup \mathcal{O}_{j+1}$ plus kj. This is because M_j contains $k + dk$ disjoint sets of size j and at least $k + dk - kd = k$ of these are disjoint from all sets in $\mathcal{O}_d \cup \mathcal{O}_{d-1} \cup \ldots \cup \mathcal{O}_{j+1}$. Hence, there is a solution amongst $\bigcup_{i \ge j} (D_i \cup M'_i)$ that is at least as good as $\mathcal{O}$ and hence is also optimal. ◀

The above lemma suggests an exact algorithm that stores the sets in $\bigcup_i (D_i \cup M'_i)$ and find the optimum solution among these sets. In particular, we construct matchings of each size greedily up to the appropriate size and store all intersecting sets. Note that since each element belongs to at most r sets, the total space is $\tilde{O}(d^2kr)$. Applying the sub-sampling framework, we have $d \le \mathrm{OPT} = O(k/\epsilon^2 \log m)$ and the approximation factor becomes $1+\epsilon$.

▶ **Theorem 10.** *There exists a randomized one-pass algorithm using $\tilde{O}(\epsilon^{-4}k^3r)$ space that finds a $1+\epsilon$ approximation to* `Max Unique Coverage` *and* `Max Coverage`.

4.2 A More Efficient $1+\epsilon$ Approximation for Maximum Coverage

In this section, we generalize the approach of Manurangsi [53] and combine that with the F_0-sketching technique to obtain a $1+\epsilon$ approximation using $\tilde{O}(\epsilon^{-3}k^2r)$ space for maximum coverage. This saves a factor k/ϵ and the generalized analysis might be of independent interest. Let $\mathrm{OPT} = \psi(\mathcal{O})$ denote the optimal coverage of the input stream.

Manurangsi [53] showed that for the maximum k-vertex cover problem, the $\Theta(k/\epsilon)$ vertices with highest degrees form a $1+\epsilon$ approximation kernel for the maximum k vertex coverage problem. That is, there exist k vertices among those that cover $(1-\epsilon)\,\mathrm{OPT}$ edges. We now consider a set system in which an element belongs to at most r sets (this can also be viewed as a hypergraph where each set corresponds to a vertex and each element corresponds to a hyperedge; we then want to find k vertices that touch as many hyperedges as possible).

We begin with the following lemma that generalizes the aforementioned result in [53]. We may assume that $m > rk/\epsilon$ since otherwise, we can store all the sets.

▶ **Lemma 11.** *Suppose $m > \lceil rk/\epsilon \rceil$. Let K be the collection of $\lceil rk/\epsilon \rceil$ sets with largest sizes (tie-broken arbitrarily). There exist k sets in K that cover $(1-\epsilon)\,\mathrm{OPT}$ elements.*

Proof. Let $\mathcal{O}$ denote the collection of k sets in some optimal solution. Let $\mathcal{O}^{in} = \mathcal{O} \cap K$ and $\mathcal{O}^{out} = \mathcal{O} \setminus K$. We consider a random subset $Z \subset K$ of size $|\mathcal{O}^{out}|$. We will show that the sets in $Z \cup \mathcal{O}^{in}$ cover $(1-\epsilon)\,\mathrm{OPT}$ elements in expectation; this implies the claim.

Let $[\mathcal{E}]$ denote the indicator variable for event $\mathcal{E}$. We rewrite

$$|\psi(Z \cup \mathcal{O}^{in})| = |\psi(\mathcal{O}^{in})| + |\psi(Z)| - |\psi(\mathcal{O}^{in}) \cap \psi(Z)| \ .$$

Furthermore, the probability that we pick a set S in K to add to Z is

$$p := \frac{|\mathcal{O}^{out}|}{|K|} \leq \frac{k}{kr/\epsilon} = \frac{\epsilon}{r} \ .$$

Next, we upper bound $\mathrm{E}\left[|\psi(\mathcal{O}^{in}) \cap \psi(Z)|\right]$. We have

$$\mathrm{E}\left[|\psi(\mathcal{O}^{in}) \cap \psi(Z)|\right] \leq \sum_{u \in \psi(\mathcal{O}^{in})} \sum_{S \in K : u \in S} \Pr\left[S \in Z\right] \leq \sum_{u \in \psi(\mathcal{O}^{in})} rp \leq |\psi(\mathcal{O}^{in})| \cdot \epsilon \ .$$

We lower bound $\mathrm{E}\left[|\psi(Z)|\right]$ as follows.

$$\begin{aligned}
\mathrm{E}\left[|\psi(Z)|\right] &\geq \mathrm{E}\left[\sum_{S \in K} \left(|S|[S \in Z] - \sum_{S' \in K \setminus \{S\}} |S \cap S'|[S \in Z \wedge S' \in Z] \right)\right] \\
&\geq \sum_{S \in K} \left(|S|p - \sum_{S' \in K \setminus \{S\}} |S \cap S'|p^2 \right) \\
&\geq \sum_{S \in K} \left(|S|p - (r-1)|S|p^2 \right) \geq p(1-pr) \sum_{S \in K} |S| \geq p(1-\epsilon) \sum_{S \in K} |S| \ . \qquad (1)
\end{aligned}$$

In the above derivation, the second inequality follows from the observation that

$$\Pr\left[S \in Z \wedge S' \in Z\right] \leq p^2 \ .$$

The third inequality is because $\sum_{S' \in K \setminus \{S\}} |S \cap S'| \leq (r-1)|S|$ since each element belongs to at most r sets.

For all $S \in K$, we must have

$$|S| \geq \frac{\sum_{Y \in \mathcal{O}^{out}} |Y|}{|\mathcal{O}^{out}|} \geq \frac{|\psi(\mathcal{O}^{out})|}{|\mathcal{O}^{out}|} \ .$$

Thus,

$$\mathrm{E}\left[|\psi(Z)|\right] \geq p\,(1-\epsilon)\,|K| \frac{|\psi(\mathcal{O}^{out})|}{|\mathcal{O}^{out}|} = p\,(1-\epsilon)\,\frac{|\psi(\mathcal{O}^{out})|}{p} = (1-\epsilon)|\psi(\mathcal{O}^{out})| \ .$$

Putting it together,

$$\mathrm{E}\left[|\psi(Z \cup \mathcal{O}^{in})|\right] \geq |\psi(\mathcal{O}^{in})| + (1-\epsilon)|\psi(\mathcal{O}^{out})| - |\psi(\mathcal{O}^{in})| \cdot \epsilon \geq (1-\epsilon)\,\mathrm{OPT} \ . \qquad \blacktriangleleft$$

With the above lemma in mind, the following algorithm's correctness is immediate.

1. Store F_0-sketches of the $\lceil kr/\epsilon \rceil$ largest sets, where the failure probability of the sketches is set to $\frac{1}{\text{poly}(n)\binom{m}{k}}$.
2. At the end of the stream, return the k sets with the largest coverage based on the estimates given by the F_0-sketches.

We restate our result as a theorem.

▶ **Theorem 12.** *There exists a randomized one-pass, $\tilde{O}(k^2r/\epsilon^3)$-space, algorithm that with high probability finds a $1+\epsilon$ approximation to* `Max Coverage`.

Obtaining a $2+\epsilon$ approximation to `Max Unique Coverage`. We note that finding the best solution to `Max Unique Coverage` in K will yield a $2+\epsilon$ approximation. This is a worse approximation than that of the previous subsection. However, we save a factor of k/ϵ in memory. Furthermore, this approach also allows us to handle streams with deletions.

To see that we get a $2+\epsilon$ approximation to `Max Unique Coverage`. Note that $g(Z \cup \mathcal{O}^{in}) \geq \frac{1}{2}\left(g(\mathcal{O}^{in}) + g(Z)\right)$. Furthermore, a similar derivation shows $\mathrm{E}\left[|\tilde{\psi}(Z)|\right] \geq (1-\epsilon)|\tilde{\psi}(\mathcal{O}^{out})|$. Specifically, in the derivation in Eq. 1, we can simply replace ψ with $\tilde{\psi}$. This gives us $g(K) \geq (1/2-\epsilon)g(\mathcal{O})$.

Extension to Insert/Delete Streams. The result can be extended to the case where sets are inserted and deleted. For the full details, see Section 6.2.

4.3 An $O(\log \min(k, r))$ Approximation for Unique Coverage

We now present an algorithm whose space does not depend on r but the result comes at the cost of increasing the approximation factor to $O(\log(\min(k, r)))$. It also has the feature that the running time is polynomial in k in addition to being polynomial in m and n.

The basic idea is as follows: We consider an existing algorithm that first finds a 2.01 approximation C to `Max Coverage`. It then finds the best solution of `Max Unique Coverage` among the sets in C.

▶ **Theorem 13.** *There exists a randomized one-pass, $\tilde{O}(k^2)$-space, algorithm that with high probability finds a $O(\log \min(k, r))$ approximation to* `Max Unique Coverage`.

Proof. From previous work [8,60], we can find a 2.01 approximation C to `Max Coverage` using $\tilde{O}(k)$ memory. Note that their algorithm maintains a collection C of k sets during the stream. Demaine et al. [25] proved that that if Q is the best solution to `Max Unique Coverage` among the sets in C, then Q is an $O(\log \min(k, r))$ approximation to `Max Unique Coverage`. In fact, they presented a polynomial time algorithm to find Q from C such that the number of uniquely covered elements is at least

$$\Omega(1/\log k) \cdot |\psi(C)| \geq \Omega(1/\log k) \cdot 1/2.01 \cdot f(M) \geq \Omega(1/\log k) \cdot g(M) \ .$$

Note that storing each set in C requires $\tilde{O}(d)$ memory. Hence, the total memory is $\tilde{O}(kd)$. Applying the sub-sampling framework, we obtain an $\tilde{O}(k^2)$ memory algorithm. ◀

4.4 Application to Parameterized Set Cover

We parameterize the set cover problem as follows. Given a set system, either A) output a set cover of size αk if $\text{OPT} \leq k$ where α the approximation factor or B) correctly declare that a set cover of size k does not exist.

▶ **Theorem 14.** *For $0 < \delta < 1$, there exists a randomized, $O(1/\delta)$-pass, $\tilde{O}(rk^2n^\delta + n)$-space, algorithm that with high probability finds a $O(1/\delta)$ approximation to the parameterized* `Set Cover` *problem.*

Proof. In each pass, we run the algorithm in Theorem 12 with parameters k and $\epsilon = 1/n^{\delta/3}$ on the remaining uncovered elements. The space use is $\tilde{O}(rk^2n^\delta + n)$. Here, we need additional $\tilde{O}(n)$ space to keep track of the remaining uncovered elements.

Note that if OPT $\leq k$, after each pass, the number of uncovered elements is reduced by a factor $1/n^{\delta/3}$. This is because if n' is the number of uncovered elements at the beginning of a pass, then after that pass, we cover all but at most $n'/n^{\delta/3}$ of those elements. After i passes, the number of remaining uncovered elements is $O(n^{1-i\delta/3})$; we therefore use at most $O(1/\delta)$ passes until we are done. At the end, we have a set cover of size $O(k/\delta)$.

If after $\omega(1/\delta)$ passes, there are still remaining uncovered elements, we declare that such a solution does not exist. ◀

Our algorithm improves upon the algorithm by Har-Peled et al. [36] that uses $\tilde{O}(mn^\delta + n)$ space for when $rk^2 \ll m$. Both algorithms yield an $O(1/\delta)$ approximation and use $O(1/\delta)$ passes.

5 Lower Bounds

5.1 Lower Bounds for Exact Solutions

As observed earlier, any exact algorithm for either the `Max Coverage` or `Max Unique Coverage` problem on an input where all sets have size d will return a matching of size k if one exists. However, by a lower bound due to Chitnis et al. [18] we know that determining if there exists a matching of size k in a single pass requires $\Omega(k^d)$ space. This immediately implies the following theorem.

▶ **Theorem 15.** *Any single-pass algorithm that solves* `Max Coverage` *or* `Max Unique Coverage` *exactly with probability at least* $9/10$ *requires* $\Omega(k^d)$ *space.*

5.2 Lower bound for a $e^{1-1/k}$ approximation

The strategy is similar to previous work on `Max Coverage` [59, 60]. However, we need to argue that the relevant probabilistic construction works for all collections of fewer than k sets since the unique coverage function is not monotone.

We make a reduction from the communication problem k-player set disjointness, denoted by `DISJ`(m, k). In this problem, there are k players where the ith player has a set $S_i \subseteq [m]$. It is promised that exactly one of the following two cases happens a) NO instance: All the sets are pairwise disjoint and b) YES instance: There is a unique element $v \in [m]$ such that $v \in S_i$ for all $i \in [k]$ and all other elements belong to at most one set. The (randomized) communication complexity (in the one-way model or the blackboard model), for some large enough constant success probability, of the above problem is $\Omega(m/k)$ even if the players may use public randomness [14]. We can assume that $|S_1 \cup S_2 \cup \ldots \cup S_k| \geq m/4$ via a padding argument.

▶ **Theorem 16.** *Any constant-pass randomized algorithm with an approximation better than $e^{1-1/k}$ to* `Max Unique Coverage` *requires $\Omega(m/k^2)$ space.*

Proof. For each $i \in [m]$, let $\mathcal{P}_i$ be a random partition of $[n]$ into k sets $V_1^i, \ldots, V_k^i$ such that an element in the universe $U = [n]$ belongs to exactly one of these sets uniformly at random. In particular, for all $i \in [m]$ and $v \in U$,

$$\Pr\left[v \in V_j^i \wedge (\forall j' \neq j, v \notin V_{j'}^i)\right] = 1/k \ .$$

The partitions are chosen independently using public randomness before receiving the input. For each player j, if $i \in S_j$, then they put V_j^i in the stream. Note that the stream consists of $\Theta(m)$ sets.

If the input is a NO instance, then for each $i \in [m]$, there is at most one set V_j^i in the stream. Therefore, for each element $v \in [n]$ and any collection of $\ell \leq k$ sets $V_{j_1}^{i_1}, \ldots, V_{j_\ell}^{i_\ell}$ in the stream,

$$\Pr\left[v \text{ is uniquely covered by } V_{j_1}^{i_1}, \ldots, V_{j_\ell}^{i_\ell}\right] = \ell/k \cdot (1-1/k)^{\ell-1} \leq \ell/k \cdot e^{-(\ell-1)/k} \ .$$

Therefore, in expectation, $\mu_\ell := \mathrm{E}\left[g(\{V_{j_1}^{i_1}, \ldots, V_{j_\ell}^{i_\ell}\})\right] \leq \ell/k \cdot e^{-(\ell-1)/k} n$. By an application of Hoeffding's inequality,

$$\begin{aligned}\Pr\left[g(\{V_{j_1}^{i_1} \cup \ldots \cup V_{j_\ell}^{i_\ell}\}) > \mu_\ell + \epsilon e^{-(k-1)/k} \cdot n\right] &\leq \exp\left(-2\epsilon^2 e^{-2(\ell-1)/k} n\right) \\ &\leq \exp\left(-\Omega(\epsilon^2 n)\right) \leq \frac{1}{m^{10k}} \ .\end{aligned}$$

The last inequality follows by letting $n = \Omega(\epsilon^{-2} k \log m)$. The following claim shows that for large k, in expectation, picking k sets is optimal in terms of unique coverage.

▶ **Lemma 17.** *The function $g(\ell) = \ell/k \cdot e^{-(\ell-1)/k} n$ is increasing in the interval $(-\infty, k]$ and decreasing in the interval $[k, +\infty)$.*

Proof. We take the partial derivative of g with respect to ℓ

$$\frac{\partial g}{\partial \ell} = \frac{e^{(1-\ell)/k}(k-\ell)}{k^2} \cdot n$$

and observe that it is non-negative if and only if $\ell \leq k$. ◀

By appealing to the union bound over all $\binom{m}{1} + \ldots + \binom{m}{k-1} + \binom{m}{k} \leq O(m^{k+1})$ possible collections $\ell \leq k$ sets, we deduce that with high probability, for all collections of $\ell \leq k$ sets $S_1, \ldots, S_\ell$,

$$\begin{aligned}g(\{S_1, \ldots, S_\ell\}) \leq \mu_\ell + \epsilon e^{-(k-1)/k} \cdot n &\leq \ell/k \cdot e^{-(\ell-1)/k} n + \epsilon e^{-(k-1)/k} \cdot n \\ &\leq (1+\epsilon) e^{-1+1/k} n \ .\end{aligned}$$

If the input is a YES instance, then clearly, the maximum k-unique coverage is n. This is because there exists i such that $i \in S_1 \cap \ldots \cap S_k$ and therefore $V_1^i, \ldots, V_k^i$ are in the stream and these sets uniquely cover all elements.

Therefore, any constant pass algorithm that returns better than a $e^{1-1/k}/(1+\epsilon)$ approximation to `Max Unique Coverage` for some large enough constant success probability implies a protocol to solve $\mathtt{DISJ}(m, k)$. Thus, $\Omega(m/k^2)$ space is required. ◀

5.3 Lower bound for $1+\epsilon$ approximation

Assadi [6] presents a $\Omega(m/\epsilon^2)$ lower bound for the space required to compute a $1+\epsilon$ approximation for `Max Coverage` when $k=2$, even when the stream is in a random order and the algorithm is permitted constant passes. This is proved via a reduction to multiple instances of the Gap-Hamming Distance problem on a hard input distribution, where an input with high maximum coverage corresponds to a YES answer for some Gap-Hamming Distance instance, and a low maximum coverage corresponds to a NO answer for all GHD instances. This hard distribution has the additional property that high maximum coverage inputs also have high maximum unique coverage, and low maximum coverage inputs have low maximum unique coverage. Therefore, the following corollary holds:

▶ **Corollary 18.** *Any constant-pass randomized algorithm with an approximation factor* $1+\epsilon$ *for* `Max Unique Coverage` *requires* $\Omega(m/\epsilon^2)$ *space.*

6 Handling Insert-Delete Streams

6.1 Proof of Theorem 7

Chitnis et al. [18] introduce a sketching primitive $\text{Sample}_{\gamma,d}$ suitable for insertion-deletion data streams which is capable of randomly sampling a diverse selection of sets. $\text{Sample}_{\gamma,d}$ first assigns a color to each element from γ colors uniformly at random. Each set in M is therefore associated with the multiset of colors assigned to its elements (a "color signature"). By maintaining an ℓ_0-sampler for all sets of each color signature, it is possible to sample one set of each color signature from $\text{Sample}_{\gamma,d}$ at the end of stream. The following lemma establishes that these sampled sets are likely to include optimal solutions for maximum coverage and maximum unique coverage.

▶ **Lemma 19.** *Let* C' *be the collection of sets sampled from* $\text{Sample}_{(2kd)^2,d}$. *Then*

$$\Pr\left[f(C')=f(M)\right]\geq 3/4 \quad \textit{and} \quad \Pr\left[g(C')=g(M)\right]\geq 3/4\ .$$

Proof. If $\text{Sample}_{(2kd)^2,d}$ assigns each element in $F(M)$ a different color, then for every set in $F(M)$ it either samples the set or one that contributes equal coverage, yielding an optimal solution. For any $i,j\in F(M)$, let $X_{i,j}=1$ if i and j receive the same color in $\text{Sample}_{(2kd)^2,d}$.

$$\Pr\left[\exists\ i,j\in F(M)\ s.t.\ color(i)=color(j)\right]\leq E\left[\sum_{i,j\in F(M)} X_{i,j}\right]<\frac{(dk)^2}{\gamma}=1/4\ . \qquad \blacktriangleleft$$

The proof for `Max Unique Coverage` is identical.

A $\text{Sample}_{(2kd)^2,d}$ sketch requires $O((kd)^{2d})$ space. Constructing $\log(k)$ $\text{Sample}_{(2kd)^2,d}$ sketches in parallel[7] guarantees that $f(C')=f(M)$ and $g(C')=g(M)$ with probability $1-1/\operatorname{poly}(k)$. This gives the desired theorem.

[7] Note that each $\text{Sample}_{(2kd)^2,d}$ sketch has a different random coloring.

6.2 Handling deletions for the algorithm in Theorem 12

We now explain how the approach using in Theorem 12 can be extended to the case where sets may be inserted and deleted. In this setting, it is not immediately obvious how to select the largest $\lceil rk/\epsilon \rceil$ sets; the approach used when sets are only inserted does not extend. Note that in this model we can set m to be the maximum number of sets that have been inserted and not deleted at any prefix of the stream rather than the total number of sets inserted/deleted.

However, we can extend the result as follows. Suppose the sketch of a set for approximating maximum (unique) coverage requires B bits; recall from Section 2.2 that $B = k\epsilon^{-2}\operatorname{polylog}(n,m)$ suffices. We can encode such a sketch of a set S as an integer $i(S) \in [2^B]$. Suppose we know that exactly $\lceil rk/\epsilon \rceil$ sets have size at least some threshold t. We will remove this assumption shortly. Consider the vector $x \in [N]$ where $N = 2^B$ that is initially 0 and then is updated by a stream of set insertions/deletions as follows:

1. When S is inserted, if $|S| \geq t$, then $x_{i(S)} \leftarrow x_{i(S)} + 1$.
2. When S is deleted, if $|S| \geq t$, then $x_{i(S)} \leftarrow x_{i(S)} - 1$.

At the end of this process $x \in \{0,1,\ldots,,m\}^{2^B}$, $\ell_1(x) = \lceil rk/\epsilon \rceil$, and reconstruct the sketches of largest ηk sets given x. Unfortunately, storing x explicitly in small space is not possible since, while we are promised that at the end of the stream $\ell_1(x) = \lceil rk/\epsilon \rceil$, during the stream it could be that x is an arbitrary binary string with m one's and this requires $\Omega(m)$ memory to store. To get around this, it is sufficient to maintain a linear sketch of x itself that support sparse recovery. For our purposes, the CountMin Sketch [22] is sufficient although other approaches are possible. The CountMin Sketch allows x to be reconstructed with probability $1-\delta$ using a sketch of size

$$O(\log N + \lceil rk/\epsilon \rceil \log(\lceil rk/\epsilon \rceil/\delta)\log m) = O(\lceil rk/\epsilon \rceil \epsilon^{-2}\operatorname{polylog}(n,m)) \ .$$

To remove the assumption that we do not know t in advance, we consider values:

$$t_0, t_1, \ldots, t_{\lceil \log_{1+\epsilon} m \rceil} \text{ where } t_i = (1+\epsilon)^i \ .$$

We define vector $x^0, x^1, \ldots \in \{0,1,\ldots,,m\}^{2^B}$ where x^i is only updated when a set of size $\leq t_i$ but $> t_{i-1}$ is inserted/deleted. Then there exists i such that $\leq \lceil rk/\epsilon \rceil$ sets have size $\leq t_{i-1}$ and the sketches of these sets can be reconstructed from $x^0, \ldots, x^{t_{i-1}}$. To ensure we have $\lceil rk/\epsilon \rceil$ sets, we may need some additional sketches corresponding to sets of size $> t_{i-1}$ and $\leq t_i$ but unfortunately there could be m such sets and we are only guaranteed recovery of x^{t_i} when it is sparse. However, if this is indeed the case we can still recover enough entries of x^{t_1} by first subsampling the entries at the appropriate rate (we can guess sampling rate $1, 1/2, 1/2^2, \ldots 1/m$) in the standard way. Note that we can keep track of $\ell_1(x^i)$ exactly for each i using $O(\log m)$ space.

7 The Subsampling Framework

Assuming we have v such that $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$. Let $h : [n] \to \{0,1\}$ be a hash function that is $\Omega(\epsilon^{-2}k\log m)$-wise independent. We run our algorithm on the subsampled universe $U' = \{u \in U : h(u) = 1\}$. Furthermore, let

$$\Pr[h(u) = 1] = p = \frac{ck\log m}{\epsilon^2 v}$$

where c is some sufficiently large constant. Let $S' = S \cap U'$ and let OPT' be the optimal unique coverage value in the subsampled set system. The following result is from McGregor and Vu [60]. We note that the proof is the same except that the indicator variables now correspond to the events that an element being uniquely covered (instead of being covered).

▶ **Lemma 20.** *With probability at least* $1 - 1/\operatorname{poly}(m)$*, we have that*

$$p\,\mathrm{OPT}(1+\epsilon) \geq \mathrm{OPT}' \geq p\,\mathrm{OPT}(1-\epsilon)$$

Furthermore, if $S_1, \ldots, S_k$ *satisfies* $g(\{S'_1, \ldots, S'_k\}) \geq p\,\mathrm{OPT}(1-\epsilon)/t$ *then*

$$g(\{S_1, \ldots, S_k\}) \geq \mathrm{OPT}(1/t - 2\epsilon)\ .$$

We could guess $v = 1, 2, 4, \ldots, n$. One of the guesses must be between $\mathrm{OPT}/2$ and OPT which means $\mathrm{OPT}' = O(\epsilon^{-2} k \log m)$. Furthermore, if we find a $1/t$ approximation on the subsampled universe, then that corresponds to a $1/t - 2\epsilon$ approximation in the original universe. We note that as long as $v \leq \mathrm{OPT}$ and h is $\Omega(\epsilon^{-2} k \log m)$-wise independent, we have (see [64], Theorem 5):

$$\begin{aligned}&\Pr\left[g(\{S'_1, \ldots, S'_\ell\}) = p \cdot g(\{S_1, \ldots, S_\ell\}) \pm \epsilon p\,\mathrm{OPT}\right]\\ &\geq 1 - \exp\left(-\Omega(k \log m)\right) \geq 1 - 1/m^{\Omega(k)}\ .\end{aligned}$$

This gives us Lemma 20 even for when $v < \mathrm{OPT}/2$. However, if $v \leq \mathrm{OPT}/2$, then OPT' may be larger than $O(\epsilon^{-2} k \log m)$, and we may use too much memory. To this end, we simply terminate those instantiations. Among the instantiations that are not terminated, we return the solution given by the smallest guess.

References

1. Alexander A. Ageev and Maxim Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. Comb. Optim.*, 8(3):307–328, 2004.
2. Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. *CoRR*, abs/1809.05082, 2018. `arXiv:1809.05082`.
3. Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013. `doi:10.1016/j.ic.2012.10.006`.
4. Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. Optimal streaming algorithms for submodular maximization with cardinality constraints. In *ICALP*, volume 168 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
5. Aris Anagnostopoulos, Luca Becchetti, Ilaria Bordino, Stefano Leonardi, Ida Mele, and Piotr Sankowski. Stochastic query covering for fast approximate document retrieval. *ACM Trans. Inf. Syst.*, 33(3):11:1–11:35, 2015.
6. Sepehr Assadi. Tight space-approximation tradeoff for the multi-pass streaming set cover problem. In *PODS*, pages 321–335. ACM, 2017.
7. Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *STOC*, pages 698–711. ACM, 2016.
8. Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. In *KDD*, pages 671–680. ACM, 2014.
9. Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM*, volume 2483 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002.
10. Édouard Bonnet, Vangelis Th. Paschos, and Florian Sikora. Parameterized exact and approximation algorithms for maximum k-set cover and related satisfiability problems. *RAIRO Theor. Informatics Appl.*, 50(3):227–240, 2016.
11. Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *ICALP (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 244–254. Springer, 2013.

12 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 263–274, 2015. doi:10.1007/978-3-662-48350-3_23.

13 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1-2):225–247, 2015.

14 Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *IEEE Conference on Computational Complexity*, pages 107–117. IEEE Computer Society, 2003.

15 Amit Chakrabarti and Anthony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *SODA*, pages 1365–1373. SIAM, 2016.

16 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2015.

17 Rajesh Chitnis and Graham Cormode. Towards a theory of parameterized streaming algorithms. In *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, pages 7:1–7:15, 2019. doi:10.4230/LIPIcs.IPEC.2019.7.

18 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *SODA*, pages 1326–1344. SIAM, 2016.

19 Rajesh Hemant Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Brief announcement: New streaming algorithms for parameterized maximal matching & beyond. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 56–58, 2015. doi:10.1145/2755573.2755618.

20 Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *SODA*, pages 1234–1251. SIAM, 2015.

21 Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Trans. Knowl. Data Eng.*, 15(3):529–540, 2003.

22 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi:10.1016/j.jalgor.2003.12.001.

23 Michael Crouch and Daniel S. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 96–104, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

24 Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 337–348, 2013. doi:10.1007/978-3-642-40450-4_29.

25 Erik D. Demaine, Uriel Feige, MohammadTaghi Hajiaghayi, and Mohammad R. Salavatipour. Combination can be hard: Approximability of the unique coverage problem. *SIAM J. Comput.*, 38(4):1464–1483, 2008.

26 Michael Dom, Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Minimum membership set covering and the consecutive ones property. In *SWAT*, volume 4059 of *Lecture Notes in Computer Science*, pages 339–350. Springer, 2006.

27 Yuval Emek and Adi Rosén. Semi-streaming set cover. *ACM Trans. Algorithms*, 13(1):6:1–6:22, 2016.

28 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM J. Discrete Math.*, 25(3):1251–1265, 2011. doi:10.1137/100801901.

29 Thomas Erlebach and Erik Jan van Leeuwen. Approximating geometric coverage problems. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 1267–1276, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347220.

30 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

31 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

32 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *STOC*, pages 1363–1374. ACM, 2020.

33 Daya Ram Gaur, Ramesh Krishnamurti, and Rajeev Kohli. Erratum to: The capacitated max k-cut problem. *Math. Program.*, 126(1):191, 2011.

34 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485, 2012. URL: http://portal.acm.org/citation.cfm?id=2095157&CFID=63838676&CFTOKEN=79617016, doi:10.1137/1.9781611973099.41.

35 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, Palo Alto, California, USA, 5-7 June, 2013*, pages 287–298, 2013. doi:10.1109/CCC.2013.37.

36 Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards tight bounds for the streaming set cover problem. In *PODS*, pages 371–383. ACM, 2016.

37 Chien-Chung Huang, Naonori Kakimura, and Yuichi Yoshida. Streaming algorithms for maximizing monotone submodular functions under a knapsack constraint. In *APPROX-RANDOM*, volume 81 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

38 Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Jonathan Ullman, Ali Vakilian, and Anak Yodpinyanee. Fractional set cover in the streaming model. In *APPROX-RANDOM*, volume 81 of *LIPIcs*, pages 12:1–12:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

39 Piotr Indyk and Ali Vakilian. Tight trade-offs for the maximum k-coverage problem in the general streaming model. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 200–217, 2019. doi:10.1145/3294052.3319691.

40 Takehiro Ito, Shin-Ichi Nakano, Yoshio Okamoto, Yota Otachi, Ryuhei Uehara, Takeaki Uno, and Yushi Uno. A 4.31-approximation for the geometric unique coverage problem on unit disks. *Theor. Comput. Sci.*, 544:14–31, 2014.

41 John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 119–133, 2019. doi:10.1145/3294052.3319706.

42 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013. doi:10.1137/1.9781611973105.121.

43 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 734–751, 2014. doi:10.1137/1.9781611973402.55.

44 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Streaming lower bounds for approximating MAX-CUT. In *SODA*, pages 1263–1282. SIAM, 2015.

45 Michael Kapralov, Sanjeev Khanna, Madhu Sudan, and Ameya Velingker. $(1 + \omega(1))$-approximation to MAX-CUT requires linear space. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1703–1722, 2017. doi:10.1137/1.9781611974782.112.

46 Michael Kapralov and Dmitry Krachun. An optimal space lower bound for approximating MAX-CUT. *CoRR*, abs/1811.10879, 2018. arXiv:1811.10879.

47 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.

48 Christian Konrad. Maximum matching in turnstile streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 840–852, 2015. doi:10.1007/978-3-662-48350-3_70.

49 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *APPROX-RANDOM*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012.

50 Christian Konrad and Adi Rosén. Approximating semi-matchings in streaming and in two-party communication. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 637–649, 2013. doi:10.1007/978-3-642-39206-1_54.

51 Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, pages 1650–1654. AAAI Press, 2007.

52 Fabian Kuhn, Pascal von Rickenbach, Roger Wattenhofer, Emo Welzl, and Aaron Zollinger. Interference in cellular networks: The minimum membership set cover problem. In *COCOON*, volume 3595 of *Lecture Notes in Computer Science*, pages 188–198. Springer, 2005.

53 Pasin Manurangsi. A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 15:1–15:21, 2019. doi:10.4230/OASIcs.SOSA.2019.15.

54 Andrew McGregor. Finding graph matchings in data streams. *APPROX-RANDOM*, pages 170–181, 2005.

55 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

56 Andrew McGregor and Sofya Vorotnikova. Planar matching in streams revisited. In *APPROX-RANDOM*, volume 60 of *LIPIcs*, pages 17:1–17:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

57 Andrew McGregor and Sofya Vorotnikova. Triangle and four cycle counting in the data stream model. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 445–456, 2020. doi:10.1145/3375395.3387652.

58 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *PODS*, pages 401–411. ACM, 2016.

59 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. In *ICDT*, volume 68 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

60 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory of Computing Systems*, pages 1–25, 2018.

61 Neeldhara Misra, Hannes Moser, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. The parameterized complexity of unique coverage and its variants. *Algorithmica*, 65(3):517–544, 2013. doi:10.1007/s00453-011-9608-0.

62 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive

data streams. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 3826–3835. PMLR, 2018.

63 Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SDM*, pages 697–708. SIAM, 2009.

64 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995.

65 Mariano Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, 2012. doi:10.1007/s00453-010-9438-5.

Diverse Data Selection under Fairness Constraints

Zafeiria Moumoulidou
College of Information and Computer Sciences, University of Massachusetts Amherst, MA, USA

Andrew McGregor
College of Information and Computer Sciences, University of Massachusetts Amherst, MA, USA

Alexandra Meliou
College of Information and Computer Sciences, University of Massachusetts Amherst, MA, USA

Abstract

Diversity is an important principle in data selection and summarization, facility location, and recommendation systems. Our work focuses on maximizing diversity in data selection, while offering fairness guarantees. In particular, we offer the first study that augments the Max-Min diversification objective with fairness constraints. More specifically, given a universe $\mathcal{U}$ of n elements that can be partitioned into m disjoint groups, we aim to retrieve a k-sized subset that maximizes the pairwise minimum distance within the set (*diversity*) and contains a pre-specified k_i number of elements from each group i (*fairness*). We show that this problem is NP-complete even in metric spaces, and we propose three novel algorithms, linear in n, that provide strong theoretical approximation guarantees for different values of m and k. Finally, we extend our algorithms and analysis to the case where groups can be overlapping.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases data selection, diversity maximization, fairness constraints, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.13

Funding This work was supported by the NSF under grants CCF-1934846, CCF-1908849, CCF-1637536, IIS-1453543, CCF-1763423, and IIS-1943971.

1 Introduction

Data is generated and collected from all aspects of human activity, in domains like commerce, medicine, and transportation, as well as scientific measurements, simulations, and environmental monitoring. However, while datasets grow large and are readily available, they are often down-sampled for various uses. This is often due to practical implications, e.g., analytics workflows may be designed, tested, and debugged over subsets of the data for efficiency reasons. Other times, machine learning applications use subsets of the data for training and testing, while applications that target human consumption, e.g., data exploration, can only display small parts of the data at a time, since human users can visually process limited information.

While data subset selection is very common, deriving *good* subsets is a non-trivial task. In this paper, we focus on two principles in data selection: *diversity* and *fairness*. Diversity and fairness are related but distinct concepts. Specifically, diversity seeks to maximize the dissimilarity of the items in a set. Intuitively, a diverse set of items selected from a dataset D represents more and different aspects of the information present in D. Prior work has suggested several diversity objectives [16, 30, 32, 43], typically defined in terms of an element-wise distance function over numerical attributes (e.g., geographic location, age). On the other hand, fairness aims to achieve some specified level of representation across different categories or groups, and is typically defined over categorical attributes (e.g., race, gender). While one could consider combining fairness and diversity into a single objective,

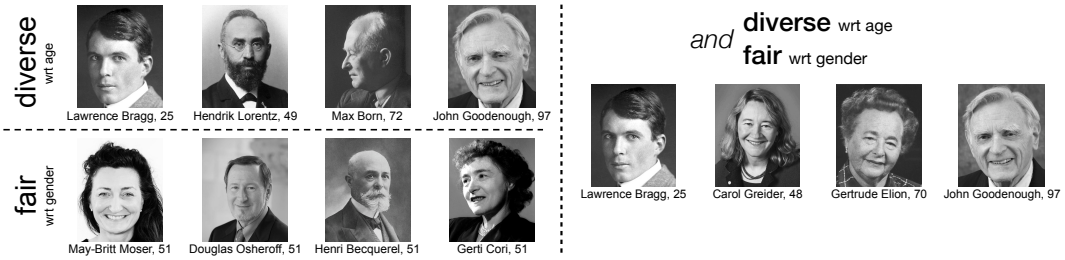


Figure 1 Three examples of selection of four items from a dataset of Nobel laureates. The first set on the left is diverse with respect to age; the second set is fair with respect to gender; the third set, on the right, is both diverse with respect to age and fair with respect to gender.

comparing numerical and categorical attributes is not straightforward, as it typically requires ad hoc decisions in discretizing numerical attributes or defining a distance function involving numeric and categorical attributes.

Figure 1 demonstrates an example of the principles of diversity and fairness in subset selection. Consider a web search query over a dataset of Nobel laureates. There are close to a thousand laureates, but the web search only serves a small number of results for human consumption. Figure 1 shows three examples of possible subsets of four items. The first subset optimizes the set's diversity with respect to the age of the laureates at the time of the award, but only contains male scientists. The second set achieves fair gender representation, but is not diverse with respect to age. The third set achieves both diversity and fairness. The concept of *fair* and *diverse* data selection is motivated by many real-world scenarios: *transportation equity* in conjunction with optimizing traditional objectives (e.g., geographic coverage) aims to design accessible transportation systems for historically disadvantaged groups [37]; formulating *teams* that represent various demographic groups while demonstrating "diversity of thought" is becoming an important hiring goal [21, 23, 31]; in *news* websites, a summary of dissimilar in context documents from different news channels minimizes redundancy and mitigates the risk of showing a polarized opinion [23].

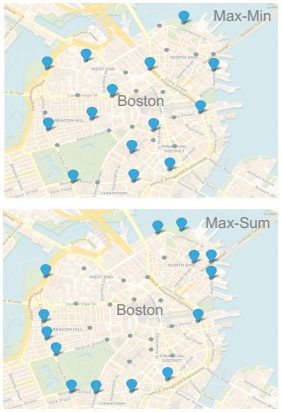
Our focus. In this paper, *our goal is to maximize diversity in data selection* with respect to numerical attributes, *while ensuring the satisfaction of fairness constraints* with respect to categorical ones. We focus on the Max-Min diversification model [23, 43, 46], which is among the most well-studied and frequently-used diversity models in the data management community. Max-Min diversification seeks to select a set of k items, such that the distance between any two items is maximized. We further express fairness as cardinality constraints: given m demographic groups, a set is *fair* if it contains a pre-specified integer number k_i of representatives from each group. This general form of cardinality constraints captures, among others, the common fairness objectives of *proportional* representation, where the sample preserves the demographic proportions of the general population, and *equal* representation, where all demographic groups are equally represented in the sample.[1] These fairness objectives have been widely studied in prior work [13, 14, 35, 45, 48, 49, 50], and fit naturally in problems of data selection where existing systems can exhibit bias with respect to sensitive attributes; e.g., a study showed that search engines tend to under-represent women in the result sets [34].

[1] Our fairness constraints are based on the definitions of *group fairness* and *statistical parity* [25]. Other definitions that focus on *individual or causal fairness* examine differences in treatment of individuals from different groups who are otherwise very similar, but these are not the focus of this work.

	Max-Min	Max-Sum
diversification	[30, 43, 46] $\frac{1}{2}$-approximation	
fair diversification (disjoint groups)	[this paper] $\frac{1}{4}$-approx. $(m = 2)$ $\frac{1}{3m-1}$-approx. $(m \geq 3)$ $\frac{1}{5}$-approx. $(m = O(1)$ and $k = o(\log n))$	[1, 9, 11] $(\frac{1}{2} - \epsilon)$-approx.
fair diversification (overlapping groups)	[this paper] $\frac{1}{4}$-approx. $(m = 2)$ $\frac{1}{3\binom{m}{\lfloor m/2 \rfloor}-1}$-approx. $(m \geq 3)$	N/A

n : # elements in the universe, m : # demographic groups, k : # elements in the data selection task

(a) Comparison with prior art.



(b) Max-Min vs Max-Sum.

Figure 2 (a) Contributions of this paper with respect to the prior art. Our work is the first to introduce fairness constraints to Max-Min diversification, and provides strong approximation results. We also contribute algorithms to the case of overlapping classes, which has not been addressed in prior work. (b) The department of transportation wants to place $k = 14$ new bike sharing stations in downtown Boston among $n = 30$ candidate locations. (Top): Max-Min selects locations that geographically *cover* downtown. (Bottom): Max-Sum selects locations on the outskirts of downtown.

We first study the problem of *fair Max-Min diversification* in the case of *non-overlapping* groups, and define the problem more formally as follows: We assume a universe of elements $\mathcal{U} = \bigcup_{i=1}^{m} \mathcal{U}_i$ partitioned into m non-overlapping groups, a metric distance function d defined for any two pairs of elements, and a set of fairness constraints $\langle k_1, k_2, \cdots, k_m \rangle$, where each k_i is a non-negative integer with $k_i \leq |\mathcal{U}_i|$. Our goal is to select a set $\mathcal{S} \subseteq \mathcal{U}$ of size $k = \sum_{i=1}^{m} k_i$, such that $|\mathcal{S} \cap \mathcal{U}_i| = k_i$ for all i, and such that the minimum distance of any two items in $\mathcal{S}$ is maximized. In this paper, we show that fair Max-Min diversification is NP-complete, and we contribute efficient algorithms with strong approximation guarantees in the case of non-overlapping groups; we further generalize our results and analysis to the case of overlapping groups. We list our contributions at the end of this section.

Contrast with prior work and related problems

Our work augments the existing literature of traditional problems that have been studied under *group fairness* constraints, such as clustering [18, 35], ranking systems [14, 48, 49, 50] and set selection [45]. We proceed to review prior work in closely-related problems and describe how our contributions augment the existing literature. (Summary shown in Figure 2a.)

Max-Min and Max-Sum diversification. The unconstrained version of Max-Min diversification is a special case of our fair variant for $m = 1$. This problem was initially studied in the operation research literature under the name *remote-edge* or *p-dispersion*, along with another popular diversity model, the *Max-Sum* or *remote-clique* model [16, 26, 30, 36, 43]. Similar formulations have also been studied in the context of obnoxious facility location on graphs [46]. While the Max-Min model aims to maximize the minimum pairwise distance in the selected set, the Max-Sum model aims to maximize the total sum of pairwise distances in a set of k items. Max-Sum, as an additive objective, is easier to analyze but tends to select points at the limits of the data space and thus it is not well-suited to applications that require more uniform coverage (see example in Figure 2b). The unconstrained diversification problems are NP-complete even in metric spaces but, for both, a greedy algorithm offers a $\frac{1}{2}-$factor approximation, that has also been shown to be tight [6, 8, 43].

(a) Example 1. **(b)** Example 2.

Figure 3 (a) An example where no optimal solution for the clustering problem is optimal for the diversity problem and vice versa. Suppose we have to pick one white point and one black point. The unique optimal solution for clustering is $\{2, 5\}$ whereas the unique optimal solution for Max-Min diversity is $\{1, 6\}$. (b) An optimal solution for the clustering problem may be arbitrarily bad for the diversity problem. Suppose we have to pick one white and one black point. Set $\{2, 3\}$ is an optimal solution for clustering but yields an arbitrarily bad approximation ratio for the diversity problem as points 2 and 3 can be arbitrarily close together.

Fair Max-Sum diversification. Abbassi et al. [1] study the *fair* Max-Sum diversification problem (assuming disjoint groups) under matroid constraints, where the retrieved subset needs to be an independent set of a matroid of size k (we discuss the correspondence between group fairness constraints and partition matroids in Appendix B). They propose a local search algorithm with a $\left(\frac{1}{2} - \epsilon\right)$-approximation guarantee. Borodin et al. [8, 9] study a bi-criteria optimization problem formulated as the sum of a submodular function and the Max-Sum diversification objective under matroid constraints. They show that the local search approach preserves the $\left(\frac{1}{2} - \epsilon\right)$-approximation guarantee. In an effort to make the state-of-the-art local search algorithms more efficient, Ceccarello et al. [11] propose algorithmic approaches for constructing core-sets with strong approximation guarantees, resulting in efficient algorithms with comparable quality to the best known local search algorithms [1, 8, 9]. A core-set is a small subset of the original data set that contains an α-approximate solution for the Max-Sum diversification problem. Cevallos et al. [15] extend the local search approach to distances of negative type and design algorithms with $O\left(1 - \frac{1}{k}\right)$-approximation and $O(nk^2 \log k)$ running time.

Fair k-center clustering. In the k-center clustering problem the objective is to select k centers such that the maximum distance of any point from its closest cluster center is minimized. Intuitively, cluster centers tend to be distributed in a way that optimizes data coverage. Thus, k-center clustering can serve as another mechanism to perform diverse data selection, albeit the optimization objective is different from Max-Min. Max-Min diversification and k-center clustering are closely related. In fact, the approximation algorithms by Gonzalez [29] for the clustering problem and by Ravi et al. [43] and Tamir [46] for Max-Min diversification, are all based on the same farthest-first traversal heuristic, and they all provide a $\frac{1}{2}$-approximation guarantee. Nonetheless, the analysis of the two algorithms is substantially different and it is not always the case that an algorithm for one problem is applicable to the other.

In recent work, Kleindessner et al. [35] introduced the fair variant of the problem, where the centers are partitioned into m different groups and the constraint of selecting k_i elements per group is enforced in the output of the process. It is easy to find examples where no optimal solution for the fair k-center problem is optimal for the Max-Min objective and vice versa (see example in Figure 3a). Furthermore, we note that an optimal solution for fair k-center clustering can be arbitrarily bad for the Max-Min objective (e.g., Figure 3b). Consequently, the two problems need to be studied independently. The fair k-center clustering problem can also be expressed by a partition matroid, for which Chen et al. [17] provide a 3-approximation algorithm with a quadratic runtime. Kleindessner et al. [35] provide a linear-time algorithm with a $\left(3 \cdot 2^{m-1} - 1\right)$-approximation, while more recent work improved

this bound to $3(1+\epsilon)$ [20], and 3-approximation [33]. In our Appendix, by adapting the ideas for fair Max-Min diversification, we design a linear-time algorithm for fair k-center clustering that also achieves a constant 3-factor approximation.

Outline of contributions: Fair Max-Min diversification. To the best of our knowledge, this paper is the first to introduce fairness constraints to Max-Min diversification. We initially focus on the case of disjoint groups, but extend our algorithms to tackle the overlapping case as well. Our work makes the following contributions.

- After some background and preliminaries, we introduce and formally define the problem of *fair* Max-Min diversification focusing on non-overlapping groups, and further discuss its complexity and approximability (Section 2). To the best of our knowledge, no prior work has studied the Max-Min diversification objective under fairness constraints. In our Appendix, we also describe how our algorithmic frameworks support any constraints that can be expressed in terms of partition matroids (Appendix B).
- We propose a swap-based greedy approximation algorithm, with linear runtime, for the case of $m = 2$, which offers a constant $\frac{1}{4}$-factor approximation guarantee (Section 3.1).
- We propose a general max-flow-based polynomial algorithm, with runtime linear in the size of the data, that offers a $\frac{1}{3m-1}$-factor approximation (Section 3.2). We also demonstrate that for constant m and *small* values for $k = o(\log n)$, we can achieve a constant $\frac{1}{5}$-approximation, also in linear time. While this bound is obviously stronger than our bound for the general case, the $\frac{1}{5}$-approximation algorithm becomes impractical as k increases (Section 3.3).
- We generalize the *fair* diversification problem to the case of overlapping groups (an element can belong to multiple demographic groups). We propose polynomial-time algorithms with $\frac{1}{4}$-factor approximation for the case of $m = 2$ and $\frac{1}{3\binom{m}{\lfloor m/2 \rfloor}-1}$-factor approximation for any m (Section 4).

2 Fair Max-Min Diversification: Background and Problem Definition

In this section, we review necessary background and preliminaries on the Max-Min diversification objective and relevant approximations. Then, we formally define the *fair* Max-Min diversification problem, which generalizes Max-Min diversification. We further characterize the hardness of the problem, and the hardness of its approximation.

2.1 Max-Min Diversification

Problem definition. Prior work has identified a range of diversity objectives to perform diverse data selection. In this work we primarily focus on the Max-Min objective, which corresponds to the minimum distance of any two items in a set $\mathcal{S}$. More formally, we assume a universe of elements $\mathcal{U}$ of size n, a positive integer $k \leq |\mathcal{U}|$ and a pseudometric distance function $d : \mathcal{U} \times \mathcal{U} \to \mathbb{R}_0^+$ that satisfies the following properties for every $u, v \in \mathcal{U}$: $d(u,u) = 0$, $d(u,v) = d(v,u)$ (symmetry), and $d(u,v) \leq d(u,w) + d(w,v)$ (triangle inequality). Then, $d(u,v)$ captures the dissimilarity of the elements $u, v \in \mathcal{U}$, and the Max-Min diversity score of a set $\mathcal{S}$ is $\text{div}(\mathcal{S}) = \min_{u,v \in \mathcal{S}, u \neq v} d(u,v)$. Max-Min diversification seeks to identify a set $\mathcal{S} \subseteq \mathcal{U}$ and $|\mathcal{S}| = k$, such that the minimum pairwise distance, $\text{div}(\mathcal{S})$, of elements in $\mathcal{S}$ is maximized.

Algorithm 1 GMM Algorithm.

Input: $\mathcal{U}$: Universe of available elements
$k \in \mathbb{Z}_0^+$
I: An initial set of elements
Output: $\mathcal{S} \subseteq \mathcal{U}$ of size k

procedure GMM($\mathcal{U}, I, k$)
  $\mathcal{S} \leftarrow \emptyset$.
  if $I = \emptyset$ **then**
    $S \leftarrow$ an arbitrarily chosen point in $\mathcal{U}$
  while $|\mathcal{S}| < k$ **do**
    $x \leftarrow \operatorname{argmax}_{u \in \mathcal{U}} \min_{s \in \mathcal{S} \cup I} d(u, s)$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{x\}$
return $\mathcal{S}$

Algorithms and approximations. This problem formulation was initially studied in the operation research literature by Ravi et al. [43] and in the context of facility location on graphs by Tamir [46]. They both show that the problem is NP-complete even in metric spaces and give a greedy algorithm, GMM, that guarantees a $\frac{1}{2}$-approximation for Max-Min diversification. Ravi et al. [43] also show that this problem cannot be approximated within a factor better than $\frac{1}{2}$ unless P=NP through a reduction from the clique problem.

The GMM approximation algorithm uses the simple and intuitive farthest-first traversal heuristic: Given a set of items $\mathcal{S}$, add the element from $\mathcal{U}$ whose minimum distance from any element in $\mathcal{S}$ is the largest. Algorithm 1 shows the pseudocode for GMM, which starts with an initial set of elements I and greedily augments it with k elements from $\mathcal{U}$. Note that the GMM algorithm, as presented by Ravi et al. [43] and Tamir [46] assumes that $I = \emptyset$; in this paper, we use the slight variant presented in Algorithm 1, which assumes that I can be non-empty. We use GMM as a building block for the algorithms we present in this paper. A naive implementation of the algorithm requires $O((|I| + k)^2 n)$ time but more efficient implementation requires $O((|I| + k)n)$ time; see, e.g., [35, 47] for details.

2.2 Fair Max-Min Diversification

Problem definition and analysis. We assume a universe of elements $\mathcal{U}$ of size n, comprising of m non-overlapping classes: $\mathcal{U} = \bigcup_{i=1}^{m} \mathcal{U}_i$; we further assume a pseudometric distance function $d : \mathcal{U} \times \mathcal{U} \to \mathbb{R}_0^+$; finally, we assume non-negative integers $\langle k_1, \ldots, k_m \rangle$, which we call *fairness constraints*. Our goal is to identify a set $\mathcal{S} \subseteq \mathcal{U}$, such that for all i, $|\mathcal{S} \cap \mathcal{U}_i| = k_i$, and the minimum distance of any two items in $\mathcal{S}$ is maximized. More formally:

$$\textsc{Fair Max-Min}: \operatorname*{maximize}_{\mathcal{S} \subseteq \mathcal{U}} \min_{\substack{u,v \in \mathcal{S} \\ u \neq v}} d(u, v)$$
$$\text{subject to } |\mathcal{S} \cap \mathcal{U}_i| = k_i, \ \forall i \in [m]$$

Intuitively, Fair Max-Min aims to derive the set with the maximum diversity score $\text{div}(\mathcal{S})$, while satisfying the fairness constraints.[2] Next, we state formally the hardness of Fair Max-Min and bound its approximability. These results follow easily from the corresponding prior results on unconstrained Max-Min diversification, as that problem reduces to Fair Max-Min for $m = 1$. We give the proof of Corollary 1 in Appendix A.

[2] A formulation of the fairness constraints with inequalities ($\geq k_i$) would be essentially equivalent: since the diversity score can only decrease as the number of selected points increases, the optimal solution would always select the minimum number of points allowed by the constraints.

▶ **Corollary 1** (Hardness and Approximability Bound). *Determining if there exists a solution to* Fair Max-Min *with diversity score* $\geq \delta$ *is NP-complete. Further, there exists no polynomial-time* α*-approximation algorithm for* Fair Max-Min *with* $\alpha > \frac{1}{2}$*, unless P=NP.*

Our contributions to this problem. To the best our knowledge, this is the first paper to augment the Max-Min diversification problem with fairness constraints. For this problem, typically m is a small constant and $k \ll n$. Therefore, when considering algorithmic complexity, we want to avoid high-order dependence on the size of the data, n. In Section 3, we provide linear-time algorithms, with respect to n, with strong approximation guarantees for this problem in the case of non-overlapping groups. In Section 4, we extend our results to design polynomial-time algorithms with strong approximation guarantees for the generalized setting of overlapping groups.

3 Approximating Diversity

In Section 2.2, we showed that the *fair* formulation for the Max-Min diversification problem is NP-hard, and cannot be approximated within a factor better than $\frac{1}{2}$. In this section, we propose three approximation algorithms for this problem, with a best overall bound of $\frac{1}{4}$ for the case of $m = 2$. For ease of exposition, in the rest of the paper we frequently refer to each of the m groups as different colors.

Our algorithms use GMM (Algorithm 1) as a building block, but adapting GMM for fair Max-Min diversification is not straightforward. We give an example of a simple and intuitive algorithm based on GMM that can lead to an arbitrarily bad result, even in the case of $m = 2$ colors. In the first phase of the algorithm, we use GMM to greedily select elements of any color until the constraints for one of them are satisfied. In the second phase, we allow GMM to greedily select the remaining elements only from the under-satisfied color. Suppose that our data consist of one white and three black elements positioned in a line as follows:

Further, consider that the fairness constraints require the selection of one white and two black elements, and that GMM first selects a black element. Regardless of which black element is selected first, the simple algorithm we described will always be forced to select elements 1 and 2 – the possible selection scenarios are: $\{1, 4, 2\}$, $\{3, 1, 2\}$, and $\{4, 1, 2\}$ – which can be arbitrarily close to one another. This example demonstrates how the choices made for one color, can lead to arbitrarily bad choices for the other color(s), and the problem gets harder as m increases.

Our algorithms employ GMM in ways that guarantee the preservation of *good* choices for all colors. We start with a swap-based algorithm that offers a $\frac{1}{4}$ approximation when $m = 2$. Then we present a flow-based algorithm with a $\frac{1}{3m-1}$ approximation when $m \geq 3$. Both algorithms run in $O(kn)$ time. Finally, we present a $\frac{1}{5}$-approximation for $m \geq 3$ that also runs in $O(kn)$, on the assumption m is constant and $k = o(\log n)$. However, the running time of this third algorithm has an additional factor that depends exponentially on k, which makes the algorithm practical only for *small* k values, e.g., for $n = 10^4$, $k \approx 10$.

3.1 Fair-and-Diverse Selection: $m = 2$

In the binary setting, the input is a set of points $\mathcal{U} = \mathcal{U}_1 \cup \mathcal{U}_2$ and two non-negative integers $\langle k_1, k_2 \rangle$ with $k_i \leq |\mathcal{U}_i|$ for all $i \in \{1, 2\}$. We want to select a set $\mathcal{S}$ with k_i elements from each $\mathcal{U}_i$ partition such that the div$(\mathcal{S})$ is maximized.

Algorithm 2 FAIR-SWAP: Fair Diversification for $m = 2$.

Input: $\mathcal{U}_1, \mathcal{U}_2$: Set of points of color 1 and 2
$k_1, k_2 \in \mathbb{Z}_0^+$
Output: k_i points in $\mathcal{U}_i$ for $i \in \{1, 2\}$
1: **procedure** FAIR-SWAP
▷Color-Blind Phase:
2: $\mathcal{S} \leftarrow \text{GMM}(\mathcal{U}, \emptyset, k_1 + k_2)$
3: $\mathcal{S}_i = \mathcal{S} \cap \mathcal{U}_i$ for $i \in \{1, 2\}$
▷Balancing Phase:
4: Set $U = \text{argmin}_i(|\mathcal{S}_i| - k_i)$ ▷Under-satisfied set
5: $O = 3 - U$ ▷Over-satisfied set
6: Compute the sets:
$E \leftarrow \text{GMM}(\mathcal{U}_U, \mathcal{S}_U, k_U - |\mathcal{S}_U|)$
$R \leftarrow \{\text{argmin}_{x \in \mathcal{S}_O} d(x, e) : e \in E\}$
return $(\mathcal{S}_U \cup E) \cup (\mathcal{S}_O \setminus R)$

Algorithm and intuition. FAIR-SWAP (Algorithm 2) has two phases; the *color-blind* and the *balancing* phase. In the *color-blind* phase, we call GMM by initializing I to the empty set so as to retrieve a set $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ of size k (line 2). If $|\mathcal{S}_1| = k_1$ and $|\mathcal{S}_2| = k_2$ then these two sets are returned. Alternatively, if one set is smaller than required, then the other set is larger than required, and we need to rebalance these sets. Let $\mathcal{S}_U$ be the set that is too small and let $\mathcal{S}_O$ be the set that is too large. The algorithm next finds $k_U - |\mathcal{S}_U|$ extra points $E \subseteq \mathcal{U}_U$ to add to $\mathcal{S}_U$ by again using the GMM algorithm, this time initialized with the set $\mathcal{S}_U$. For each point in E we then remove the closest point in $\mathcal{S}_O$ (line 6). In this way we add $k_U - |\mathcal{S}_U|$ points to $\mathcal{S}_U$ and remove $k_U - |\mathcal{S}_U|$ points from $\mathcal{S}_O$. After this rebalancing the size of $\mathcal{S}_U$ is $|\mathcal{S}_U| + (k_U - |\mathcal{S}_U|) = k_U$ and the size of $\mathcal{S}_O$ is $|\mathcal{S}_O| - (k_U - |\mathcal{S}_U|) = k - k_U = k_O$ as required. Note that sets E and R will be empty if the sets are already balanced after the color blind phase and thus the set $\mathcal{S}$ will not be altered by the balancing phase.

Running-time analysis. The running time of FAIR-SWAP (Algorithm 2) is $O(kn)$. In the color-blind phase of the algorithm we run GMM on $\mathcal{U}$ with $I = \emptyset$ and this takes $O(kn)$ time. Then in the balancing phase, computing the extra points E via the GMM algorithm takes $O(kn)$ time and computing R takes $O(k^2)$ time since there are fewer than k points in E and at most k points in $\mathcal{S}_O$.

Approximation-factor analysis. Let $\mathcal{S}^*$ be the set of k points in $\mathcal{U}$ that maximize the diversity when there are no fairness constraints. Let $\ell^* = \text{div}(\mathcal{S}^*)$. Let $\mathcal{F}^* = \mathcal{F}_1^* \cup \mathcal{F}_2^*$ be the set of k points in $\mathcal{U}$ that maximize the diversity subject to the constraint that for each $i \in \{1, 2\}$, k_i points are chosen of color i. Let $\ell^*_{\text{fair}} = \text{div}(\mathcal{F}^*)$ and note that $\ell^* \geq \ell^*_{\text{fair}}$.

We first argue that $\text{div}(\mathcal{S}) \geq \ell^*/2 \geq \ell^*_{\text{fair}}/2$. This follows because, by the triangle inequality, there is at most one point in $\mathcal{S}^*$ that is distance $< \ell^*/2$ from each point in $\mathcal{S}$; otherwise two points in $\mathcal{S}^*$ would be $< \ell^*$ apart and this contradicts the fact $\text{div}(\mathcal{S}^*) = \ell^*$. Hence, while the GMM algorithm has picked $< k$ elements, there exists at least one element in $\mathcal{S}^*$ that can be selected that is distance $\geq \ell^*/2$ from all the points already selected. Since the algorithm picks the next point farthest away from the points already chosen, the next point is at least $\ell^*/2$ from the existing points.

Next we argue that $\text{div}(\mathcal{S}_U \cup E) \geq \ell^*_{\text{fair}}/2$. To show this, first observe that, $\text{div}(\mathcal{S}_U) \geq \text{div}(\mathcal{S}) \geq \ell^*_{\text{fair}}/2$. Next consider the points added to E by GMM. By the triangle inequality there is at most one point in $\mathcal{F}_U^*$ that is distance $< \ell^*_{\text{fair}}/2$ from each point in $\mathcal{S}_U \cup E$. Hence,

while GMM has picked $< k_U - |\mathcal{S}_U|$ elements, there exists at least one element that can be selected that is distance $\geq \ell^*_{\text{fair}}/2$ from the points already selected. Since the algorithm picks the next point farthest away from the points already chosen, the next point is at least $\ell^*_{\text{fair}}/2$ from the existing points. Thus, we can guarantee that $d(x, y) \geq \ell^*_{\text{fair}}/2$ for all pairs of points $x, y \in \mathcal{S}_U \cup E \cup \mathcal{S}_O$ except potentially when $x \in E$ and $y \in \mathcal{S}_O$.

To handle this case, for each $x \in E$ we remove the closest point in $\mathcal{S}_O$. Note that by an application of the triangle inequality and the fact that $\text{div}(\mathcal{S}_O) \geq \ell^*_{\text{fair}}/2$, for each $x \in E$ there can be at most one point $y \in \mathcal{S}_O$ such that $d(x, y) < \ell^*_{\text{fair}}/4$. Hence, after the removal of the closest points the distance between all pairs is $\geq \ell^*_{\text{fair}}/4$ as required. We summarize the analysis of this section as follows:

▶ **Theorem 2.** *Fair-Swap (Algorithm 2) is a $1/4$-approximation algorithm for the fair diversification problem when $m = 2$ that runs in time $O(kn)$.*

Connections to prior art. The idea of balancing has also been successfully applied to matroid optimization settings subject to fairness constraints [19], and to the red-blue matching problem [39]. However, our objective function cannot be expressed by a matroid (or an intersection of matroids), and thus the approaches of prior work are not applicable to our setting. Further, the algorithms and analysis are distinct for these problems; Fair-Swap builds upon GMM while the algorithms designed in [19] employ the Edmonds algorithm for finding a maximum independent set.

3.2 Fair-and-Diverse Selection: $m \geq 3$

Basic algorithm. We start by presenting a basic algorithm that takes as input a guess γ for the optimum fair diversity. If this guess is greater than the optimum fair diversity then the algorithm may abort, but if the algorithm does not abort, it will return a fair diversity at least $\gamma/(3m - 1)$.

Algorithm and intuition. The approach of Fair-Flow (Algorithm 3) is to construct disjoint sets of points $C_1, C_2, \ldots$ such that, if γ is at most the optimal fair diversity, it is possible to find sets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ of sizes $k_1, \ldots, k_m$ such that each C_i contains at most one point from $\mathcal{S}_1 \cup \ldots \cup \mathcal{S}_m$. If we can construct $C_1, C_2, \ldots$ such that for any $x \in C_i$ and $y \in C_j$, then $d(x, y) \geq d_2$ for some value d_2 then we have $\text{div}(\mathcal{S}_1 \cup \ldots \cup \mathcal{S}_m) \geq d_2$. Furthermore, because the sets $C_1, C_2, \ldots$ are disjoint it is possible to find sets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ with the required property via a reduction to network flow (noting that the optimal flow in a network with integer capacities is always integral). See the algorithm for the precise reduction and see Figure 4 for an example.

The way we construct each $C_1, C_2, \ldots$ is to first run GMM on each color class i and use this to identify at most k points Z_i of color i such that $\text{div}(Z_i) \geq d_1$ for some value d_1 to be determined. We then partition $\bigcup_i Z_i$ into the disjoint groups $C_1, C_2, \ldots$ where the partition satisfies the property that any two points $x, y \in \bigcup_i Z_i$ such that $d(x, y) < d_2$ are in the same group. Note that x, z will end up in the same group if there exists y such that $d(x, y) < d_2$ and $d(y, z) < d_2$; more generally two points can end up in the same group because of a chain of points where each adjacent pair of points are close. However, in the analysis, we will show that these chains cannot be too long and, for appropriately chosen d_1 and d_2, any two points in C_j are distance $< d_1$ from each other. In the analysis, this will enable us to argue that if γ is at most the optimal fair diversity, it is possible to find the required sets $\mathcal{S}_1, \ldots, \mathcal{S}_m$.

Algorithm 3 FAIR-FLOW: Fair Diversification for $m \geq 3$.

Input: $\mathcal{U}_1, \ldots, \mathcal{U}_m$: Universe of available elements
$k_1, \ldots, k_m \in \mathbb{Z}_0^+$
$\gamma \in \mathbb{R}$: A guess of the optimum fair diversity
Output: k_i points in $\mathcal{U}_i$ for $i \in [m]$

procedure FAIR-FLOW
  for $i \in [m]$ **do**
    $Y_i \leftarrow \text{GMM}(\mathcal{U}_i, \emptyset, \sum_i k_i)$
  $Z_i \leftarrow$ maximal prefix of Y_i such that all points in Z_i are $\geq d_1 = \frac{m\gamma}{3m-1}$ apart.
  Construct undirected graph G_Z with nodes $Z = \bigcup_i Z_i$ and edges (z_1, z_2), if $d(z_1, z_2) < d_2 = \frac{\gamma}{3m-1}$.
  $C_1, C_2, \ldots C_t \leftarrow$ Connected components of G_Z.
  ▷Construct flow graph
  Construct directed graph $G = (V, E)$ where

$$\begin{aligned} V &= \{a, u_1, \ldots, u_m, v_1, \ldots, v_t, b\} \\ E &= \{(a, u_i) \text{ with capacity } k_i : i \in [m]\} \\ &\quad \cup \{(v_j, b) \text{ with capacity } 1 : j \in [t]\} \\ &\quad \cup \{(u_i, v_j) \text{ with capacity } 1 : |Z_i \cap C_j| \geq 1\} \end{aligned}$$

  Compute max a-b flow.
  if flow size $< k = \sum_i k_i$ **then return** $\emptyset$ ▷Abort
  else ▷max flow is k
    $\forall (u_i, v_j)$ with flow add a node in C_j with color i to $\mathcal{S}$.
  return $\mathcal{S}$

Analysis of basic algorithm. We need a preliminary lemma that argues that all the points in the same connected component are close together.

▶ **Lemma 3.** *For all connected components C_j, $\forall x, y \in C_j : \ d(x, y) < (m-1)d_2$ and C_j does not contain any two points of the same color.*

Proof. Consider two points $x, y \in C_j$ and let the length of a shortest unweighted path $P_{x,y}$ between x and y in the graph be ℓ. If $\ell \leq m-1$ then $d(x, y) < (m-1)d_2$ as required. If $\ell \geq m$ then there exists two points on this path (including end points) that have the same color and this will lead to a contradiction. Consider the subpath $P_{x',y'} \subset P_{x,y}$ where x' and y' have the same color i and all internal nodes have distinct colors. Then the length of $P_{x',y'}$ is strictly less than $md_2 = m\gamma/(3m-1) = d_1$. But this contradicts $d(x', y') \geq d_1$ for all points in Z_i. ◀

The next theorem establishes that when the algorithm does not abort, the solution returned has diversity at least $\gamma/(3m-1)$ and that it never aborts if the guess γ is at most the optimum diversity.

▶ **Theorem 4.** *Let ℓ^*_{fair} be the optimum diversity. If $\gamma \leq \ell^*_{fair}$ then the algorithm returns a set of points of the required colors that are each $\geq \gamma/(3m-1)$ apart. If $\gamma > \ell^*_{fair}$ then the algorithm either aborts or returns a set of points of the required colors that are each $\geq d_2 = \gamma/(3m-1)$ apart.*

Proof. Note that if the algorithm does not abort then all points are $\geq \gamma/(3m-1)$ apart since any two points in different connected components are $\geq \gamma/(3m-1)$ apart.

Hence, it remains to argue that if $\gamma \leq \ell^*_{\text{fair}}$ then the algorithm does not abort. To argue this, we will construct a flow of size k in the network instance. And to do this it suffices to identify k_i connected components including a point from Z_i for each i, such that the resulting set of $k_1 + k_2 + \ldots + k_m$ connected components are all distinct. To do this, we start by defining a node u_i to be critical if $|Z_i| < k$ and non-critical otherwise. Let $O_i \subset \mathcal{U}_i$ be the set of k_i points in the optimum solution. For $x \in \mathcal{U}_i$, let $f(x)$ be the closest point Z_i to x. If

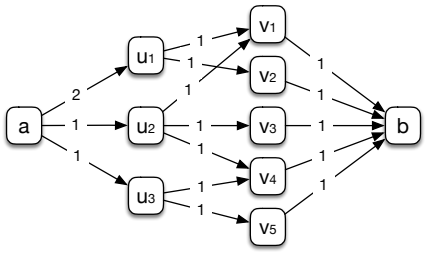


Figure 4 The graph construction in Algorithm 3 (line 7) corresponding to $m = 3$, $k_1 = 2, k_2 = 1, k_3 = 1$. Points of color 1 are contained in C_1 and C_2. Points of color 2 are contained in C_1, C_3, and C_4. Points of color 3 are contained in C_4 and C_5. Note there is an a-b flow of size $k_1 + k_2 + k_3$ iff it is possible to pick at most one point from each C_j while still picking at most k_i points of color i for each $i \in [m]$.

u_i is critical, then note that $d(x, f(x)) < d_1$. Note that for all points $x, y \in \cup_{i:\text{critical}} f(O_i)$, $d(x, y) > \ell^*_{\text{fair}} - 2d_1 \geq \gamma - 2\gamma m/(3m-1) = (m-1)d_2$ and hence, by Lemma 3, this implies that all points in $\bigcup_{i:\text{critical}} f(O_i)$ are in different connected components.

We then consider each non-critical node u_i in turn. Since u_i was non-critical and each connected component has at most one point in each Z_i, there are k connected components that include a point in Z_i. At most $k - k_i$ need to be used to pick points of other classes and hence at least $k - (k - k_i) = k_i$ remain. ◀

Final algorithm. Our final algorithm is based on binary searching for a "good" guess γ for the optimum diversity ℓ^*_{fair} where each guess can be evaluated using the basic algorithm above. The goal is to find a guess that is close to ℓ^*_{fair} or larger such that the algorithm does not abort. There are two natural ways to do this; which is best depends on parameters of the data set.

Binary-searching over continuous range: For the first approach, note that $\ell^*_{\text{fair}} \in [d_{min}, d_{max}]$ where $d_{\min} = \min_{x,y \in \mathcal{U}: x \neq y} d(x, y)$, and $d_{\max} = \max_{x,y \in \mathcal{U}: x \neq y} d(x, y)$. Hence, there exists a guess $\gamma = (1+\epsilon)^i d_{\min}$ for some $i \in \{0, 1, 2, \ldots, \lceil \log_{1+\epsilon} R \rceil\}$ where $R := d_{\max}/d_{\min}$ such that $\ell^*_{\text{fair}}/(1+\epsilon) \leq \gamma \leq \ell^*_{\text{fair}}$. Note that for this guess, the algorithm returns a $(3m-1)(1+\epsilon)$ approximation. We can find this guess (or an even better guess, i.e., a $\gamma > \ell^*_{\text{fair}}$ for which the algorithm does not abort) via a binary search over the $1 + \lceil \log_{1+\epsilon} R \rceil$ possible guesses. The number of trials required is $O(\log(1 + \lceil \log_{1+\epsilon} R \rceil) = O(\log(\epsilon^{-1}) + \log\log R)$.

Binary-searching over discrete set: For the second approach we note that after the algorithm's initial step (which did not depend on the guess γ) there are only km points and hence at most $\binom{km}{2}$ distinct distances between remaining points. Hence, it suffices to only consider guesses γ such that d_1 or d_2 corresponds to one of these $O(k^2m^2)$ values. We can sort these values in $O(k^2m^2 \log km)$ time and then binary search over this range to find a good guess using $O(\log km)$ trials.

Final diversification result. Our main theorem of this section follows by combining the binary search over a discrete set approach with the basic algorithm.

▶ **Theorem 5.** *There is a $\frac{1}{3m-1}$-approximation algorithm for the fair diversity problem that runs in time $O(kn + k^2m^2 \log(km))$.*

Proof. The time to construct $Y_1, \ldots, Y_m$ is $O(kn)$. We then need to sort the $O(k^2m^2)$ distances amongst these points. This takes $O(k^2m^2 \log(km))$ time. The time to construct and solve the flow instance is $O(k^2m^2)$ since the flow instance has $O(km)$ nodes and $O(km)$ edges [40, 41]. Note that the binary search requires us to construct and solve $O(\log(km))$ flow instances. Hence the total running time is as claimed. ◀

Algorithm 4 FAIR-GMM: Fair Diversification for small k.

Input: $\mathcal{U}_1, \ldots, \mathcal{U}_m$: Universe of available elements
$k_1, \ldots, k_m \in \mathbb{Z}_0^+$
Output: k_i points in $\mathcal{U}_i$ for $i \in [m]$
1: **procedure** FAIR-GMM
2: **for** $i \in [m]$ **do** $Y_i \leftarrow \text{GMM}(\mathcal{U}_i, \emptyset, \sum_i k_i)$
3: By exhaustive search, find the sets $\mathcal{S}_i \subseteq Y_i$ for $i \in [m]$ such that $|\mathcal{S}_i| = k_i$ and $\text{div}(\mathcal{S}_1 \cup \ldots \cup \mathcal{S}_m)$ is maximized.

If we used the binary search over a continuous range approach, the running time would by $O(kn + k^2m^2(\log \epsilon^{-1} + \log\log d_{\max}/d_{\min}))$ and the approximation ratio would be $\frac{1}{(3m-1)(1+\epsilon)}$.

3.3 Fair-and-Diverse Selection: Small k, m

In this section, we present a simple algorithm that has the advantage of achieving a better approximation ratio than the algorithm in the previous section. The downside of the algorithm is that the running time is exponential in k, specifically, $O(kn + k^2(em)^k)$. However, when $m = O(1)$ and $k = o(\log n)$ the dominating term in the running time is $O(kn)$, as in the case of the algorithms from the previous sections.

Algorithm and intuition. The basic approach of FAIR-GMM (Algorithm 4) is to first select k points (or less if there are fewer than k points of a particular color) of each color via the GMM algorithm. The resulting subset $\bigcup_i Y_i$ has at most km points and this is significantly smaller than the original set of points assuming k and m are much smaller than n. Hence, it is feasible to solve the problem via exhaustive search on the subset of points. In the analysis, we will be able to show that the optimal fair diversity amongst the subset of points is at least $1/5$ of the optimal fair diversity amongst $\bigcup_i \mathcal{U}_i$.

Analysis. To prove the approximate factor we need to show that the optimal solution amongst the subset of points selected in step one has diversity that is not significantly smaller than the optimal diversity of the original set of points. To show this the basic idea is that for each i, the set Y_i will contain at least one point near every color i point in the optimal solution or will contain k points such that even if we remove any set of $k - k_i$ points to make space for points of other colors, the remaining set of k_i points of color i still has sufficiently high diversity.

▶ **Theorem 6.** *Algorithm 4 returns a $\frac{1}{5}$-approximation and the running time is $O(kn + k^2(em)^k)$. Note that this is $O(kn)$ when $k = o(\log n)$ and $m = O(1)$.*

Proof. For the running time, note that Step 1 can be implemented in $O(kn)$ time. For Step 2, note that there are at most km points in $Y_1, Y_2, \ldots Y_m$ so a brute force algorithm needs to consider at most $\binom{km}{k} \leq (em)^k$ sets of points and computing the min distance for each takes $O(k^2)$ time. Note that this is $o(n)$ assuming $k = o(\log n)$ and m is constant.

For the approximation ratio, it suffices to argue that if ℓ^*_{fair} is the optimum value then there exists a set of points amongst $Y_1 \cup \ldots \cup Y_m$ with the required colors that are $\ell^*_{\text{fair}}/5$ apart. Let Z_i be the maximal prefix of Y_i such that all points at points are $\geq 2\ell^*_{\text{fair}}/5$ apart. For each $x \in \mathcal{U}_i$, let $f(x)$ be the closest point in Z_i. Call i critical if $|Z_i| < k$. Note that if i is critical, then $d(x, f(x)) < 2\ell^*_{\text{fair}}/5$. Let O_i be the optimal set of color i points and consider the subsets $\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_m$ of points in $Z_1, Z_2, \ldots Z_m$ defined as follows:

- For all i that are critical, let $\mathcal{S}_i = f(O_i)$ and let $D = \cup_{i:\text{critical}} \mathcal{S}_i$. Note that $\text{div}(D) > \ell^*_{\text{fair}} - 4\ell^*_{\text{fair}}/5 = \ell^*_{\text{fair}}/5$.

Figure 5 An example with $m = 2$ overlapping classes, with $|\mathcal{U}_1| = 3$ and $|\mathcal{U}_2| = 4$, where (a) the fairness constraints can be satisfied with fewer than k elements and (b) a class has to be overrepresented to satisfy the fairness constraints for all classes. Suppose we have to pick two white and one black element ($k = 3$). A feasible solution consists of two bi-colored elements, thus fewer than k, in which the black class is represented by two and not just one element.

- For each j that is not critical: Remove all points in Z_j that are distance $< \ell^*_{\text{fair}}/5$ from a point in D. Note that at most one point in Z_j is $< \ell^*_{\text{fair}}/5$ from each point in D because points in Z_j are $\geq 2\ell^*_{\text{fair}}/5$ apart. Hence, at most $|D|$ points are removed from Z_j.
- Process the non-critical j in arbitrary order: Pick k_j points $\mathcal{S}_j$ arbitrarily from Z_j. Remove all points from Z that are distance $< \ell^*_{\text{fair}}/5$ from a point in $\mathcal{S}_j$. This removes at most k_j points from each Z_i. Note that when we process j there are at least $k - (\sum_{i:\mathcal{S}_i \text{ defined so far}} k_i) \geq k - (k - k_j) = k_j$ points in Z_j.

Note $\text{div}(\bigcup_i \mathcal{S}_i) \geq \ell^*_{\text{fair}}/5$ and this implies the claimed approximation factor. ◀

4 Generalizing to Overlapping Groups

In this section, we show how we can extend our algorithmic framework to allow the elements in the universe $\mathcal{U}$ to belong to multiple classes, e.g., an individual may belong to multiple demographic groups such as multiple races, or combinations of race, gender, and other sensitive demographics. First, we formally define the problem and show how our Fair-Swap and Fair-Flow algorithms can be adapted to support this generalized setting.

We assume a universe of elements $\mathcal{U}$ comprising of m *possibly* overlapping classes $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m$, a pseudometric distance function $d : \mathcal{U} \times \mathcal{U} \to \mathbb{R}^+_0$ and a set of fairness constraints $\langle k_1, \ldots, k_m \rangle$ where each k_i is a non-negative integer with $k_i \leq |\mathcal{U}_i|$. Our goal is to identify a set $\mathcal{S} \subseteq \mathcal{U}$ to satisfy the fairness constraints such that the minimum distance of any two items in $\mathcal{S}$ is maximized.

It will be convenient to introduce some additional notation. For any $L \subset [m]$, define $X_L = \left(\bigcap_{i \in L} \mathcal{U}_i\right) \cap \left(\bigcup_{j \notin L} \mathcal{U}_j\right)$. That is, X_L consists of all elements exactly in the classes of L and no others. Note that if we select an element in X_L it contributes to helping satisfy $|L|$ of the fairness constraints. Hence, it may be possible to satisfy all the constraints by picking fewer than $k_1 + \ldots + k_m$ elements. Further, a feasible solution may require more than k_i elements for class i (example in Figure 5). Formally, we define the problem as follows:

$$\text{Fair}^+ \text{ Max-Min}: \underset{\mathcal{S} \subseteq \mathcal{U}}{\text{maximize}} \quad \min_{\substack{u,v \in \mathcal{S} \\ u \neq v}} d(u, v)$$
$$\text{subject to } |\mathcal{S} \cap \mathcal{U}_i| \geq k_i, \ \forall i \in [m]$$

4.1 Fair-and-Diverse Selection (Overlaps): $m = 2$

In the binary setting, the input is a set of points $\mathcal{U}$ that comprises of $m = 2$ overlapping classes; $\mathcal{U}_1 = X_{\{1\}} \cup X_{\{1,2\}}$ and $\mathcal{U}_2 = X_{\{2\}} \cup X_{\{1,2\}}$. We design a swap-based algorithm, with 1/4-approximation guarantee, which uses the idea of binary searching over a discrete set of guesses for the optimum fair diversity, denoted as ℓ^*_{fair}.

Algorithm and intuition. The Fair$^+$-Swap algorithm (Algorithm 5) takes as input a guess γ for the optimum fair diversity. We show that if $\gamma \leq \ell^*_{\text{fair}}$, we can always find enough points to construct a fair set $\mathcal{S} = \mathcal{S}_{\{1\}} \cup \mathcal{S}_{\{2\}} \cup \mathcal{S}_{\{1,2\}}$ with $\text{div}(\mathcal{S}) \geq \gamma/4$ (where $\mathcal{S}_L = \mathcal{S} \cap X_L$).

The algorithm first finds as many points as possible in $X_{\{1,2\}}$ and are at least $\frac{\gamma}{4}$ apart from each other. Let $\mathcal{S}_{\{1,2\}}$ be the resulting set, with a total of t points. Note that to satisfy the fairness constraints, we need to add $k_i - t$ points for each class i in $\{1,2\}$. The algorithm proceeds to remove all points in $\mathcal{U}$ that are closer than $\frac{\gamma}{4}$ from any point in $\mathcal{S}_{\{1,2\}}$. It is easy to see that all remaining points, $\mathcal{S}^+$, can only belong to one class, i.e., $\mathcal{S}^+ \cap X_{\{1,2\}} = \emptyset$ (because all points that did not make it to $\mathcal{S}_{\{1,2\}}$ have to be closer than $\frac{\gamma}{4}$ from some point in $\mathcal{S}_{\{1,2\}}$). Since $\mathcal{S}^+$ does not have overlapping classes, we can execute Fair-Swap (Algorithm 2) on it to select a set with $k_i - t$ points for each class i in $\{1,2\}$. In our analysis, we show that $\mathcal{S}^+$ contains at least $k_1 - t$ and $k_2 - t$ points from $X_{\{1\}}$ and $X_{\{2\}}$ that are $\geq \gamma$ apart from each other. Thus, the Fair$^+$-Swap algorithm will produce a set of points that are at least $\gamma/4$ apart from each other.

▶ **Theorem 7.** *Fair$^+$-Swap (Algorithm 5) is a polynomial-time algorithm with* $1/4$*-approximation guarantee for the fair diversification problem with* $m = 2$ *overlapping classes.*

We provide the pseudocode for the Fair$^+$-Swap algorithm (described above) and the proof for Theorem 7 in Appendix A.

4.2 Fair-and-Diverse Selection (Overlaps): $m \geq 3$

The algorithm in this section is an extension of Fair-Flow (Algorithm 3); the previous algorithm did not apply in the case when classes could overlap whereas the new algorithm will. Throughout this section, it will be convenient to use the following notation: $M := \binom{m}{\lfloor m/2 \rfloor}$. The approximation factor for the algorithm designed in this section will be $3M - 1$ in contrast to the $3m - 1$ approximation for the non-overlapping case. Note that for $m = 2, 3, 4, 5$ we have $M = 2, 3, 6, 10$, i.e., when the number of classes is small, M is still relatively small.

There are two main steps that need to be changed in the overlapping case: 1) defining a subset Z of the elements that will be considered and 2) determining how many points to use that appear in multiple classes. We discuss each in turn.

Defining Z. Recall that the first main part of Fair-Flow (Algorithm 3) was to select a subset of points of each color such that all points in each subset was a certain distance apart. When there are overlapping classes, we need to revisit how this is done. Motivated by the fact that an element in $X_{L'}$ contributes to at least as many fairness constraints as an element in X_L if $L \subset L'$, when we select a subset of points in $\mathcal{U}_i$ we want to prioritize points that are also in other classes. For example, for $m = 3$ we have: (1) $\mathcal{U}_1 = X_{\{1\}} \cup X_{\{1,2\}} \cup X_{\{1,3\}} \cup X_{\{1,2,3\}}$, (2) $\mathcal{U}_2 = X_{\{2\}} \cup X_{\{1,2\}} \cup X_{\{2,3\}} \cup X_{\{1,2,3\}}$, and (3) $\mathcal{U}_3 = X_{\{3\}} \cup X_{\{1,3\}} \cup X_{\{2,3\}} \cup X_{\{1,2,3\}}$.

Consistent with "prioritizing points" in multiple classes, we construct subsets of $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$ by first constructing a maximal subset $Z_{\{1,2,3\}} \subset X_{\{1,2,3\}}$ such that the pairwise distance of all points is at least d_1. We then define a maximal subset $Z_{\{1,3\}} \subset X_{\{1,3\}}$ such that every point is at least d_1 from each other point in $Z_{\{1,3\}}$ and from points in $Z_{\{1,2,3\}}$. We construct $Z_{\{1,2\}}$ and $Z_{\{2,3\}}$ similarly. Finally $Z_{\{1\}}$ is a maximal subset of $X_{\{1\}}$ such that every point is at least d_1 from each other point in $Z_{\{1\}}$ and from every point in $Z_{\{1,2\}} \cup Z_{\{1,3\}} \cup Z_{\{1,2,3\}}$. Lines 3–5 in Algorithm 6 (given in the Appendix) generalize this process to arbitrary m.

Note that we ensure the property that all points in Z_L are at least d_1 far from each other and from any point in $\bigcup_{L':L\subset L'} Z_{L'}$ but the subset of elements picked from $\mathcal{U}_1$, i.e., $Z_{\{1\}} \cup Z_{\{1,2\}} \cup Z_{\{1,3\}} \cup Z_{\{1,2,3\}} \subset \mathcal{U}_1$, no longer satisfies the condition that they are all at least d_1 far from one another. In particular, there may exist points $x \in Z_L$ and $y \in Z_{L'}$ such that

$d(x,y) < d_1$ if neither L or L' is a subset of the other.[3] A natural question, and an issue that will arise in our analysis is how many sets can there be such that no set is a subset of another. Fortunately, the following classic result in extremal combinatorics resolves this question.

▶ **Lemma 8** (Sperner's Lemma). *A collection of sets is called an* anti-chain *if none of the sets is a subset of another set. If all sets are subsets of $[m]$ then the maximum size of such a collection is $M = \binom{m}{\lfloor m/2 \rfloor}$.*

Next, recall that Fair-Flow (Algorithm 3) then constructs a graph G_Z where the nodes are the selected points and there are edges between points if their distance is $< d_2$. The new algorithm proceeds similarly but with new parameters: $d_1 \leftarrow \frac{M\gamma}{3M-1}$, and $d_2 \leftarrow \frac{\gamma}{3M-1}$. With this setting of the parameters and appealing to Lemma 8 we prove an upper bound on the distance between any two points in the same connected components (proof in Appendix A):

▶ **Lemma 9.** *For all connected components C_j, $\forall x, y \in C_j : \; d(x,y) < (M-1)\, d_2$, and C_j does not contain any two points a, b such that $a \in X_L$ and $b \in X_{L'}$ where $L \subset L'$.*

Guessing how much to exploit points in multiple classes. So far we have (1) discussed how to select the subset Z of input points and (2) partitioned Z such that we have some upper bound on the distance between any two points in the same partition. In the non-overlapping case, we could then argue it suffices to pick at most one point in each partition and adding this point to the output set $\mathcal{S}$ would increment $|\mathcal{S} \cap \mathcal{U}_i|$ for exactly one value $i \in [m]$. In the overlapping case, however, we may need to pick a point in a partition that is in multiple classes and would increment $|\mathcal{S} \cap \mathcal{U}_i|$ for multiple values of i.

To get the reduction to network flow to generalize to the non-overlapping case we need to guess values c_L for every non-empty set $L \subset [m]$ and require that we find at least c_L points in $\cap_{i \in L} \mathcal{U}_i$ such that the $\sum_{L \subseteq [m]} c_L$ points returned are distinct. The fact the points need to be distinct allows the reduction to go through. Note that to satisfy the fairness requirements we need that $\sum_{L: i \in L} c_L \geq k_i$ for each i.

▶ **Example 10.** Suppose we require $k_1 = 2$ points from $\mathcal{U}_1$ and $k_2 = 2$ points from $\mathcal{U}_2$. Then the guess $c_{\{1\}} = 2$ and $c_{\{2\}} = 2$ would correspond to picking at least four distinct points, at least two from $\mathcal{U}_1$ and at least two from $\mathcal{U}_2$. In contrast, the guess $c_{\{1\}} = c_{\{2\}} = 1$, and $c_{\{1,2\}} = 1$ would correspond to picking at least three distinct points where at least one comes from each of sets $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_1 \cap \mathcal{U}_2$ respectively.

There are at most $k^{2^m - 1 - m}$ possible guesses[4] to try for the values and at least one is feasible since the optimal solution corresponds to some set of guesses. With a feasible set of guesses, we then essentially treat all sets $L \subseteq [m]$ as colors although when we need to pick c_L points of color L, it will suffice to pick points with color L' if L is a subset of L'.

The next theorem establishes that when the algorithm does not abort, the solution returned has diversity at least $\gamma/(3M-1)$ and that it never aborts if the guess γ is at most the optimum diversity.

[3] This is a generalization of the case when there was no-overlap. In that case there could exist $x \in Z_i$ and $y \in Z_j$ such that $d(x,y) < d_1$.

[4] Recall that we typically consider m to be a small constant. A bound of $k^{2^m - 1}$ is immediate because there at most $2^m - 1$ quantities. A slightly tighter bound follows by noting that c_L for all singleton sets L is implied once the other values are chosen.

▶ **Theorem 11.** *Let ℓ^*_{fair} be the optimum diversity. If $\gamma \leq \ell^*_{fair}$ then the algorithm returns a set of points of the required colors that are each $\geq \gamma/(3M-1)$ apart. If $\gamma > \ell^*_{fair}$ then the algorithm either aborts or returns a set of points of the required colors that are each $\geq d_2 = \gamma/(3M-1)$ apart.*

We provide the proof of Theorem 11 in Appendix A. The rest of the algorithm and analysis follows similarly as Algorithm 3, where we binary search for γ in either a continuous or discrete space. The running time is increased by a factor of k^{2^m-m-1} because of the need to guess the values $\{c_L\}_{L\subset[m]}$; thus FAIR$^+$-FLOW is a polynomial-time algorithm with a $\frac{1}{3\binom{m}{\lfloor m/2\rfloor}-1}$-approximation guarantee.

5 Related Work

Diversity is an important principle in data selection and summarization, facility location, recommendation systems and web search. The diversity models that have been proposed in the literature can be organized into three main categories, (1) the distance-based models where the goal is to minimize the *similarity* of the elements within a set, (2) the coverage-based models where there exists a predetermined number of categories and the aim is to maximize the *coverage* of these categories [4, 38] and (3) the novelty-based models that are defined so as to minimize the *redundancy* of the elements shown to the user [10]. For further information, we refer the reader to the related surveys [23, 24].

Max-Min and Max-Sum diversification are two of the most well studied distance-based models [16, 28, 30, 43], and there exist efficient algorithms with strong approximation guarantees for the unconstrained version of the problems in the offline setting (discussed in Sections 1 and 2). The problem of diversity maximization has also been studied in the streaming and distributed settings, where (composable) core-sets were shown to be a useful theoretical tool [3, 12, 32], and more recently in the sliding window setting [7]. A separate line of work focuses on designing efficient indexing schemes for result diversification [2, 22, 47]; this direction is orthogonal to our work, and it is not clear how to extend existing indexing schemes for fair Max-Min diversification.

There is relatively little prior work on constrained diversification. The closest to our work is fair Max-Sum diversification (discussed in Section 1) and fair k-center clustering (discussed in Section 1 and Appendix C). To the best of our knowledge, our work is the first to augment the traditional Max-Min objective with fairness constraints.

Prior work has also combined fairness with the determinant measure of diversity [13]. That work models fairness constraints the same way as we do, but their algorithmic framework is entirely different. There, data is represented as vectors, and at each iteration the algorithm identifies the item that is most orthogonal to the current vector, which gets updated with the new item's projection. The limitation of this method is that it can only work in high-dimensional data (e.g., it would not work at all on one-dimensional data). Other work on diverse set selection focused on satisfying fairness constraints while optimizing an additive utility [45]. These methods do not apply to our setting as Max-Min is not additive. Prior work has also examined the satisfaction of fairness constraints or preferences in specialized settings, such as rankings [14, 48, 49]. Work in this domain focuses on specifying and measuring fairness and augmenting ranking algorithms with fairness considerations. Related work on diverse top-k results focuses on returning search results by a combined measure of relevance and dissimilarity to results already produced [5, 42].

Our fairness constraints are based on the definitions of *group fairness* and *statistical parity* [25]. We do not pick a particular definition of fairness, and do not place particular restrictions on the values and distribution of $\langle k_1, \ldots, k_m \rangle$. This model can express equal and proportional representation, as well as any other distribution. There are other, non-parity-based definitions of fairness that fall outside our framework. For example, *individual or causal fairness* [27] examine differences in treatment of individuals from different groups who are otherwise very similar, but these are not the focus of this work.

6 Summary and Future Directions

In this paper, we focused on the problem of diverse data selection under fairness constraints. To the best of our knowledge, our work is the first to introduce fairness constraints to Max-Min diversification. We studied both cases of disjoint and overlapping groups and proposed novel polynomial algorithms with strong approximation guarantees. For the case of disjoint groups, our algorithms have linear running time with respect to the size of the data. Overall, our work augments in significant ways the existing literature of traditional problems that have been studied under group fairness constraints. We discuss here some possible directions that extend our work through the exploration of problem variants, or intuitions towards improvement of the known algorithms and bounds.

Improved bounds. An interesting open question is whether an $\frac{1}{2}$ approximation for Fair Max-Min is possible, as is the case for Max-Min and fair Max-Sum diversification. In Appendix B, we discuss the correspondence between fairness constraints and partition matroids. It is possible that results relevant to matroids can be exploited to improve the algorithms and bounds for the Fair Max-Min problem.

Extending the swap algorithm to the general case. Our Fair-Swap algorithm provides a better bound compared to our Fair-Flow algorithm for the case of $m = 2$ ($\frac{1}{4}$ and $\frac{1}{5}$ respectively). This indicates the possibility that the swap algorithm, if extended to the general case, could perhaps result in a better bound than Fair-Flow.

Problem variants. Our algorithms aim to approximate the diversity score of the optimal solution to Fair Max-Min, while guaranteeing the satisfaction of the fairness constraints. A possible problem variant could explore the relaxation of the fairness constraints, and seek to minimize their violation while guaranteeing a diversity score at least as good as the solution to unconstrained Max-Min diversification. Another interesting future direction is to study the fair variant of other diversity objectives proposed in the literature [16, 32], for which there are currently no known results.

References

1 Zeinab Abbassi, Vahab S. Mirrokni, and Mayur Thakur. Diversity maximization under matroid constraints. In *KDD '13*, pages 32–40, 2013.

2 Pankaj K. Agarwal, Stavros Sintos, and Alex Steiger. Efficient indexes for diverse top-k range queries. In *PODS '20*, page 213–227, 2020.

3 Sepideh Aghamolaei, Majid Farhadi, and Hamid Zarrabi-Zadeh. Diversity maximization via composable coresets. In *CCCG*, 2015.

4 Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *WSDM '09*, page 5–14, 2009.

5 Albert Angel and Nick Koudas. Efficient diversity-aware search. In *SIGMOD '11*, page 781–792, 2011.

6 Aditya Bhaskara, Mehrdad Ghadiri, Vahab Mirrokni, and Ola Svensson. Linear relaxations for finding diverse elements in metric spaces. In *NIPS'16*, page 4105–4113, 2016.

7 Michele Borassi, Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. Better sliding window algorithms to maximize subadditive and diversity objectives. In *PODS '19*, page 254–268, 2019.

8 Allan Borodin, Aadhar Jain, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions, and dynamic updates. *ACM Trans. Algorithms*, 2017.

9 Allan Borodin, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS '12*, pages 155–166, 2012.

10 Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98*, page 335–336, 1998.

11 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Fast coreset-based diversity maximization under matroid constraints. In *WSDM '18*, pages 81–89, 2018.

12 Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. Mapreduce and streaming algorithms for diversity maximization in metric spaces of bounded doubling dimension. *Proc. VLDB Endow.*, page 469–480, 2017.

13 Elisa Celis, Vijay Keswani, Damian Straszak, Amit Deshpande, Tarun Kathuria, and Nisheeth Vishnoi. Fair and diverse DPP-based data summarization. In *ICML '2018*, pages 716–725, 2018.

14 L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi. Ranking with fairness constraints. In *ICALP*, 2017.

15 Alfonso Cevallos, Friedrich Eisenbrand, and Rico Zenklusen. Local search for max-sum diversification. In *SODA '17*, page 130–142, 2017.

16 Barun Chandra and Magnús M Halldórsson. Approximation algorithms for dispersion problems. *J. Algorithms*, pages 438–465, 2001.

17 Danny Z. Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, pages 27–52, 2016.

18 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *NIPS'17*, pages 5036–5044, 2017.

19 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvtiskii. Matroids, matchings, and fairness. In *Proceedings of Machine Learning Research*, PMLR '19, 2019.

20 Ashish Chiplunkar, Sagar Kale, and Sivaramakrishnan Natarajan Ramamoorthy. How to solve fair k-center in massive data models. In *ICML 2020*, pages 1877–1886, 2020.

21 Anesa "Nes" Diaz-Uda, Carmen Medina, and Beth Schill. Diversity's new frontier: Diversity of thought and the future of the workforce. Deloitte Insights, 2013. URL: `https://www2.deloitte.com/us/en/insights/topics/talent/diversitys-new-frontier.html`.

22 M. Drosou and E. Pitoura. Diverse set selection over dynamic data. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1102–1116, 2014.

23 Marina Drosou, H.V. Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. Diversity in big data: A review. *Big Data*, 5:73–84, 2017.

24 Marina Drosou and Evaggelia Pitoura. Search result diversification. *SIGMOD Rec.*, pages 41–47, 2010.

25 Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *ITCS '12*, pages 214–226, 2012.

26 Erhan Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.

27 Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: Testing software for discrimination. In *ESEC/FSE '17*, pages 498–510, 2017.

28 Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW '09*, page 381–390, 2009.

29 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

30 Refael Hassin, Shlomi Rubinstein, and Arie Tamir. Approximation algorithms for maximum dispersion. *Oper. Res. Lett.*, 21(3):133–137, October 1997.

31 Vivian Hunt, Dennis Layton, and Sara Prince. Why diversity matters. McKinsey & Company, 2015. URL: `https://www.mckinsey.com/business-functions/organization/our-insights/why-diversity-matters`.

32 Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *PODS '14*, page 100–108, 2014.

33 Matthew Jones, Huy Nguyen, and Thy Nguyen. Fair k-centers via maximum matching. In *ICML 2020*, pages 4940–4949, 2020.

34 Matthew Kay, Cynthia Matuszek, and Sean A. Munson. Unequal representation and gender stereotypes in image search results for occupations. In *CHI '15*, page 3819–3828, 2015.

35 Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *ICML '19*, volume 97, pages 3448–3457, 09–15 June 2019.

36 Michael J. Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987.

37 Todd Litman. Evaluating transportation equity: Guidance for incorporating distributional impacts in transportation planning, 2020.

38 Sean A. Munson, Daniel Xiaodan Zhou, and Paul Resnick. Sidelines: An algorithm for increasing diversity in news and opinion aggregators. In *ICWSM*, 2009.

39 Christos Nomikos, Aris Pagourtzis, and Stathis Zachos. Randomized and approximation algorithms for blue-red matching, 2007. `doi:10.1007/978-3-540-74456-6_63`.

40 James B. Orlin. Max flows in o(nm) time, or better. In *STOC'13*, pages 765–774, 2013. `doi:10.1145/2488608.2488705`.

41 James B. Orlin and Xiao-Yue Gong. A fast max flow algorithm. *CoRR*, abs/1910.04848, 2019. `arXiv:1910.04848`.

42 Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Diversifying top-k results. *Proc. VLDB Endow.*, 5(11):1124–1135, July 2012.

43 S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Oper. Res.*, 42(2):299–310, April 1994.

44 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

45 Julia Stoyanovich, Ke Yang, and H. V. Jagadish. Online set selection with fairness and diversity constraints. In *EDBT*, 2018.

46 Arie Tamir. Obnoxious facility location on graphs. *SIAM J. Discrete Math.*, 4:550–567, November 1991.

47 Yue Wang, Alexandra Meliou, and Gerome Miklau. Rc-index: Diversifying answers to range queries. *Proc. VLDB Endow.*, 11(7):773–786, 2018.

48 Ke Yang, Vasilis Gkatzelis, and Julia Stoyanovich. Balanced ranking with diversity constraints. In *IJCAI'19*, pages 6035–6042, 2019.

49 Ke Yang and Julia Stoyanovich. Measuring fairness in ranked outputs. In *SSDBM '17*, 2017.

50 Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. Fa*ir: A fair top-k ranking algorithm. In *CIKM '17*, pages 1569–1578, 2017.

Appendix

A Additional Algorithms and Proofs

In this section, we prove the hardness and approximation bound results for Fair Max-Min, which are formally stated in Corollary 1. Further, we provide proofs for the theoretical results described in Section 4. We also provide the pseudocode for the Fair$^+$-Swap algorithm (Algorithm 5), described in Section 4.1, and the pseudocode for the Fair$^+$-Flow algorithm (Algorithm 6), described in Section 4.2.

Proof of Corollary 1. First, we show that Fair Max-Min is an NP-complete problem. The problem is clearly in NP: If we are given a solution $\mathcal{S}$, we can verify that it satisfies the fairness constraints and compute its diversity score in polynomial time. The unconstrained version of Max-Min diversification is NP-complete [43, 46], and it is a special case of our problem for $m = 1$. Since any instance of Max-Min diversification can be reduced to an instance of Fair Max-Min with $m = 1$, then Fair Max-Min is also NP-complete.

Subsequently, we show that Fair Max-Min cannot be approximated with an approximation factor better than $\frac{1}{2}$. Suppose that there exists a polynomial algorithm that approximates the diversity score of the optimal solution to Fair Max-Min by a factor of $\alpha > \frac{1}{2}$. Then, this algorithm could also solve the unconstrained Max-Min diversification problem with approximation factor α. However, Ravi et al. [43] have shown that unconstrained Max-Min diversification cannot be approximated within a factor better than $\frac{1}{2}$, through a reduction from the clique problem. Therefore, it is not possible for such an algorithm to exist. ◀

Proof of Theorem 7. Let $O = O_{\{1\}} \cup O_{\{2\}} \cup O_{\{1,2\}}$ be the optimal set that maximizes diversity and satisfies the fairness constraints. Let $\ell^*_{\text{fair}} = \text{div}(O)$, which implies that $d(o_1, o_2) \geq \ell^*_{\text{fair}}$ for any pair of optimal elements $o_1, o_2 \in O$. We will show that for any guess $\gamma \leq \ell^*_{\text{fair}}$, Algorithm 5 returns a set $\mathcal{S} = \mathcal{S}_{\{1\}} \cup \mathcal{S}_{\{2\}} \cup \mathcal{S}_{\{1,2\}}$ with $\text{div}(\mathcal{S}) \geq \gamma/4$.

First, note that by the definition of $\mathcal{S}_{\{1,2\}}$ set, it holds that $\text{div}(\mathcal{S}_{\{1,2\}}) \geq \gamma/4$. Next, notice that the $\mathcal{S}^-$ set in line 3 of Algorithm 5 consists of all the points in $X_{\{1,2\}}$, and all single-colored points $< \gamma/4$ apart from some point in $\mathcal{S}_{\{1,2\}}$. As a result, we know that: (1) all the points remaining in $\mathcal{S}^+ = \mathcal{U} \setminus \mathcal{S}^-$ are greater or equal than $\gamma/4$ apart from all the points in $\mathcal{S}_{\{1,2\}}$, and (2) $\mathcal{S}^+$ only contains single-colored points (if there were any bi-colored elements $\geq \gamma/4$ apart from the points in $\mathcal{S}_{\{1,2\}}$, they would have been added to $\mathcal{S}_{\{1,2\}}$).

We further express $\mathcal{S}^+ = \mathcal{S}^+_{\{1\}} \cup \mathcal{S}^+_{\{2\}}$ with $\mathcal{S}^+_{\{i\}} \subseteq X_{\{i\}}$ for $i \in \{1, 2\}$ and $t = |\mathcal{S}_{\{1,2\}}|$. We argue that for any guess $\gamma \leq \ell^*_{\text{fair}}$, $\mathcal{S}^+$ contains at least $k_i - t$ elements for $i \in \{1, 2\}$ that are $\geq \gamma$ apart. Thus, Fair-Swap will be able to find a set of points to satisfy the fairness constraints that are at least $\gamma/4$ apart.

Define $c^-_{\{1\}}, c^-_{\{2\}}, c^-_{\{1,2\}}$ to be the number of optimal points in $O_{\{1\}}, O_{\{2\}}$ and $O_{\{1,2\}}$ present in $\mathcal{S}^-$ and notice that $c^-_{\{1\}} + c^-_{\{2\}} + c^-_{\{1,2\}} \leq t$. This holds because at most one optimal point can be $< \gamma/4$ from a point in $\mathcal{S}_{\{1,2\}}$. Suppose that there exist a pair of optimal points $o_1, o_2 \in O$, and a point $x \in \mathcal{S}_{\{1,2\}}$ such that $d(o_1, x) < \gamma/4$ and $d(o_2, x) < \gamma/4$. Then we derive a contradiction by applying the triangle inequality as: $d(o_1, o_2) \leq d(o_1, x) + d(x, o_2) < \gamma/2 < \ell^*_{\text{fair}}/2$. Consequently, it now follows that $\mathcal{S}^+$ contains at least $k_1 - c^-_{\{1\}} - c^-_{\{1,2\}} \geq k_1 - t$ optimal points of $O_{\{1\}}$, and $k_2 - c^-_{\{2\}} - c^-_{\{1,2\}} \geq k_2 - t$ of $O_{\{2\}}$, which by definition of O are greater or equal than γ apart.

So Fair-Swap will be able to find a set $\mathcal{S}_{\{1\}} \subseteq \mathcal{S}^+_{\{1\}}$ and $\mathcal{S}_{\{2\}} \subseteq \mathcal{S}^+_{\{2\}}$ with the required number of elements such that $\text{div}(\mathcal{S}_{\{1\}} \cup \mathcal{S}_{\{2\}}) \geq \gamma/4$. Thus, we get that $\text{div}(\mathcal{S}) \geq \gamma/4$. If we perform a binary search over all the pairwise distances of the points in $\mathcal{U}$, we will find a guess $\gamma = \ell^*_{\text{fair}}$, which implies the claimed approximation factor for Fair$^+$-Swap. ◀

Algorithm 5 FAIR$^+$-SWAP: Overlapping classes for $m = 2$.

Input: $\mathcal{U}_1, \mathcal{U}_2$: Universe of available elements
$\gamma \in \mathbb{R}$: A guess on the optimum fair diversity
$k_1, k_2 \in \mathbb{Z}_0^+$
Output: at least k_i points in $\mathcal{U}_i$ for $i \in \{1, 2\}$
1: **procedure** FAIR$^+$-SWAP
2: $\mathcal{S}_{\{1,2\}} \leftarrow$ maximal subset of $X_{\{1,2\}}$ with all points $\geq \gamma/4$ apart
3: $\mathcal{S}^- \leftarrow$ all the points in $\mathcal{U}$ that $< \gamma/4$ apart from a point in $\mathcal{S}_{\{1,2\}}$
4: $\mathcal{S}^+ \leftarrow \mathcal{U} \setminus \mathcal{S}^-$ ▷$\mathcal{S}^+ = \mathcal{S}^+_{\{1\}} \cup \mathcal{S}^+_{\{2\}} \subseteq X_{\{1\}} \cup X_{\{2\}}$
▷Select the missing points to satisfy the constraints:
5: Set $t = |\mathcal{S}_{\{1,2\}}|$
6: **if** $|\mathcal{S}^+ \cap \mathcal{U}_i| \geq k_i - t$ for $i \in \{1, 2\}$ **then**
7: $\mathcal{S}_{\{1\}} \cup \mathcal{S}_{\{2\}} \leftarrow$FAIR-SWAP$(\mathcal{S}^+, k_1 - t, k_2 - t)$
8: $\mathcal{S} \leftarrow \mathcal{S}_{\{1\}} \cup \mathcal{S}_{\{2\}} \cup \mathcal{S}_{\{1,2\}}$
9: **else**
10: $\mathcal{S} \leftarrow \emptyset$ ▷Abort
return $\mathcal{S}$

Proof of Lemma 9. Consider two points $x, y \in C_j$ and let the length of a shortest unweighted path $P_{x,y}$ between x and y in the graph be ℓ. If $\ell \leq M - 1$ then $d(x, y) < (M - 1)d_2$ as required. If $\ell \geq M$ then by Lemma 8, there must exist two points on this path (including end points) in X_L and $X_{L'}$ such that L and L' are *comparable*, i.e., L is a subset of L' or vice versa and this will lead to a contradiction. Consider the subpath $P_{x',y'} \subset P_{x,y}$ such that $x' \in X_L$ and $y' \in X_{L'}$ for some comparable L and L'. If the internal nodes are $x_1, x_2, \ldots$ and these belong to sets $X_{L_1}, X_{L_2}, \ldots$ then by definition of x' and y', the collection of sets $\{L_1, L_2, \ldots, L'\}$ is an anti-chain and hence the size of this collection is at most M by Lemma 8. Hence, the length of the path between x' and y' is also at most M and therefore $d(x', y') < Md_2 = d_1$. But this contradicts $d(x', y') \geq d_1$ because $x' \in X_L$ and $y' \in X_{L'}$ where L and L' are comparable. ◀

Proof of Theorem 11. Note that if the algorithm does not abort then all points are $\geq \gamma/(3M - 1)$ apart since any two points in different connected components are $\geq \gamma/(3M - 1)$ apart. Hence, it remains to argue that if $\gamma \leq \ell^*_{\text{fair}}$ then the algorithm does not abort.

To argue this, we will show it is possible to construct a flow of size $\sum c_L$. And to do this it suffices to, for each $L \subset [m]$, identify c_L different connected components that each include a point from $\cap_{i \in L} \mathcal{U}_i$. Let $O = \bigcup_{L \subset [m]} O_L$ be an optimal solution where $O_L = O \cap X_L$ and let $c_L = |O_L|$. We will henceforth consider the iteration of the algorithm which guessed this set of $\{c_L\}_{L \subset [m]}$ values.

For every point $x \in O$, let $f(x)$ be the closest point in Z where for all i, $x \in \mathcal{U}_i \Rightarrow f(x) \in \mathcal{U}_i$. Note that this requirement ensures that if x is replaced by $f(x)$ then all the fairness constraints are still satisfied. By construction of Z, $d(x, f(x)) < d_1$. Hence, for any $x, y \in O$, $d(x, y) > \ell^*_{\text{fair}} - 2d_1 \geq \gamma - 2\gamma M/(3M - 1) = (M - 1)d_2$, and hence, by Lemma 9, this implies that all points in $f(O)$ are in different connected components. This implies that there exist connected components with the necessary requirements. ◀

B Fairness as a Partition Matroid

While the focus of our work is on fairness constraints in particular, our results apply in general to any type of constraints that can be expressed in terms of a partition matroid. We provide a brief overview of the matroid definition and show that fairness constraints can be expressed as a partition matroid.

Algorithm 6 FAIR$^+$-FLOW: Overlapping classes for $m \geq 3$.

Input: $\mathcal{U}_1, \ldots, \mathcal{U}_m$: Universe of available elements
$c_L \in \mathbb{Z}^+$ for all $L \subset [m]$: A guess of the flow distribution
$\gamma \in \mathbb{R}$: A guess of the optimum fair diversity
$k_1, \ldots, k_m \in \mathbb{Z}_0^+$
Output: at least k_i points in $\mathcal{U}_i$ for $i \in [m]$

procedure FAIR$^+$-FLOW
  Define $d_1 \leftarrow \frac{M\gamma}{3M-1}$ and $d_2 \leftarrow \frac{\gamma}{3M-1}$
  $Z_{[m]} \leftarrow$ maximal subset of $X_{[m]}$ with all points $\geq d_1$ apart
  for $t = m-1, m-2, \ldots, 1$ **do**
    for all sets of L of size t **do**
      $Z_L \leftarrow$ maximal subset of X_L s.t. each point in Z_L is $\geq d_1$ from every other point in

$$Z_L \cup \bigcup_{L' \in [m]: |L'| \geq t+1, L \subset L'} Z_{L'}$$

  $G_Z \leftarrow$ undirected graph with nodes $Z = \bigcup_{L \subset [m]} Z_L$ and edges (z_1, z_2) if $d(z_1, z_2) < d_2$
  $C_1, C_2, \ldots C_t \leftarrow$ Connected components of G_Z
  ▷Construct flow graph
  Construct directed graph $G = (V, E)$ where

$$\begin{aligned} V &= \{a, v_1, \ldots, v_t, b\} \cup \bigcup_{L \subset [m]: |L| > 0} \{u_L\} \\ E &= \{(a, u_L) \text{ with capacity } c_L : \text{non-empty } L \subset [m]\} \\ &\quad \cup \{(v_j, b) \text{ with capacity } 1 : j \in [t]\} \\ &\quad \cup \{(u_L, v_j) \text{ with capacity } 1 : |Z_L \cap C_j| \geq 1\} \end{aligned}$$

  Compute max a-b flow.
  if flow size $< \sum_{L \subset [m]} c_L$ **then return** $\emptyset$ ▷Abort
  else
    $\forall (u_L, v_j)$ with flow add a node in $C_j \in (\cap_{i \in L} \mathcal{U}_i)$ to $\mathcal{S}$
  return $\mathcal{S}$

▶ **Definition 12.** *A matroid $\mathcal{M}$ is a pair $(\mathcal{E}, \mathcal{I})$ where $\mathcal{E}$ is a ground set and $\mathcal{I}$ is a collection of subsets of E (called independent sets). All the independent sets in $\mathcal{I}$ satisfy the following properties:*

- *If $\mathcal{A} \in \mathcal{I}$, then for every subset $\mathcal{B} \subseteq \mathcal{A}$, $\mathcal{B} \in \mathcal{I}$. (Hereditary property)*
- *If $\mathcal{A}, \mathcal{B} \in \mathcal{I}$ with $|\mathcal{A}| > |\mathcal{B}|$, then $\exists e \in \mathcal{A} \setminus \mathcal{B}$ such that $\mathcal{B} \cup \{e\} \in \mathcal{I}$. (Exchange property)*

A maximal independent set in $\mathcal{I}$ (also called a basis for a matroid) is a set for which there is no element outside of the set that can be added so that the set still remains independent. All maximal independent sets of a matroid have equal cardinality which is also called the rank of the matroid, rank($\mathcal{M}$).

▶ **Definition 13.** *A matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ is a partition matroid if $\mathcal{E}$ can be decomposed into m disjoint sets $\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_m$ and $\mathcal{I}$ is defined as $\mathcal{I} = \{S \subseteq \mathcal{E} : |S \cap \mathcal{E}_i| \leq k_i \ \forall \ i \in [m]\}$.*

Note that a maximal independent set (or a basis) for a partition matroid is an independent set that satisfies all the cardinality constraints with equality. For further information on matroids, we refer the interested reader to [44]. Based on the definitions above, in FAIR MAX-MIN the ground set is the universe of elements $\mathcal{U} = \bigcup_{i=1}^{m} \mathcal{U}_i$. Then FAIR MAX-MIN can be expressed as searching for the maximal independent set of the partition matroid defined over $\mathcal{U}$ that maximizes the Max-Min diversity function.

C Results on fair k-center clustering

In this paper, our primary focus has been on fair diversification based on the Max-Min objective. However, as we discussed in Section 1, fair k-center clustering is a closely-related problem. In this section, we formally define k-center clustering, introduce its fair variant and discuss the known approximation results. We then explore how algorithms and intuitions from our work on fair Max-Min diversification can be adapted towards the fair k-center clustering problem to achieve a constant factor 3-approximation.

The k-center and fair k-center clustering problems. The objective of k-center clustering is to identify k cluster centers, such that the maximum distance of any point in the universe of elements $\mathcal{U}$ from its closest cluster center is minimized. This maximum distance is referred to as the *clustering radius*. More formally, given a distance metric d, k-center clustering is expressed by the following minimization problem: $\underset{\mathcal{S}\subseteq\mathcal{U},|\mathcal{S}|=k}{\text{minimize}} \max_{u\in\mathcal{U}} d(u,\mathcal{S})$, where $d(u,\mathcal{S}) = \min_{s\in\mathcal{S}} d(u,s)$. Note that this objective does not preclude cluster centers from being close to each other, and in fact an optimal solution to k-center clustering could be arbitrarily bad for Max-Min diversification.

Algorithms and approximations. Just like Max-Min diversification, k-center clustering is NP-complete. The greedy approximation algorithm proposed by Gonzalez [29] is essentially equivalent to GMM (Algorithm 1) and provides a 2-approximation with linear running time.

Notably, there is recent work that augments the problem with fairness constraints [35]: Given m non-overlapping classes in $\mathcal{U} = \cup_{i=1}^{m}\mathcal{U}_i$ and non-negative integers $\langle k_1,\ldots,k_m\rangle$, the goal is to derive a set of cluster centers $\mathcal{S}$, such that $|\mathcal{S}\cap\mathcal{U}_i| = k_i$. The fair k-center clustering problem can also be expressed by a partition matroid, for which Chen et al. [17] provide a 3-approximation with a quadratic runtime. Kleindessner et al. [35] provide a linear-time 5-approximation algorithm for the case of two classes ($m = 2$), and a linear-time $\big(3\cdot 2^{m-1}-1\big)$-approximation for the general case, a result recently improved to $3(1+\epsilon)$ by Chiplunkar et al. [20] and to 3-approximation by Jones et al. [33]. In Section C.1, we adapt the flow algorithm for fair Max-Min diversification, and provide a linear-time 3-approximation for fair k-center clustering. (noting that the three results were derived independently.)

C.1 Fair k-center clustering

We show how we can adapt our Fair-Flow algorithm (Algorithm 3) and design a constant factor 3-approximation for fair k-center clustering with linear running time.

Basic algorithm. We start by presenting a basic algorithm that takes as input a guess γ for the optimum fair clustering radius. If this guess is less than the optimum fair clustering radius r^*_{fair} then the algorithm may abort but otherwise it will return a fair clustering with radius at most 3γ.

Algorithm and intuition. The basic idea behind Fair-Flow-Clust (Algorithm 7) is to construct a set of points $Y = \{y_1,\ldots,y_t\}$ where all distances between these points are $> 2\gamma$ apart and all points not in this set are $\leq 2\gamma$ from some point in Y; this can be done via the GMM algorithm (lines 2 and 7). The fact that each pair is $> 2\gamma$ apart implies that any k-center clustering, fair or otherwise, with covering radius $\leq\gamma$ has the property that at least one center must be within a distance γ from each y_i and that no center is within distance γ of two points y_i, y_j since, by appealing to the triangle inequality, this would violate the fact that $d(y_i,y_j) > 2\gamma$.

Algorithm 7 FAIR-FLOW-CLUST: Fair k-Center Clustering.

Input: $\mathcal{U}_1, \ldots, \mathcal{U}_m$: Universe of available elements
$k_1, \ldots, k_m \in \mathbb{Z}^+$
$\gamma \in \mathbb{R}$: A guess of optimum fair clustering radius.
Output: k_i points in $\mathcal{U}_i$ for each $i \in [m]$

1: **procedure** FAIR-FLOW-CLUST
2: $Y = \{y_1, \ldots, y_{k+1}\} \leftarrow \text{GMM}(\mathcal{U}, \emptyset, k+1)$
3: **for** $j \in [k]$ **do**
4: $D_j \leftarrow \{\text{argmin}_{x \in \mathcal{U}_i} d(x, y_j) : i \in [m]\}$
5: **if** $d(y_{k+1}, \{y_1, \ldots, y_k\}) > 2\gamma$ **then return** $\emptyset$ ▷Abort
6: **else**
7: $Y = \{y_1, \ldots, y_t\}$ with minimum $t \leq k$ such that
$d(y_{t+1}, \{y_1, \ldots, y_t\}) \leq 2\gamma$
8: **for** $j \in [t]$ **do**
9: $C_j \leftarrow \{x \in D_j : d(x, y_j) \leq \gamma\}$
▷Construct flow graph
10: Construct directed graph $G = (V, E)$ where

$$\begin{aligned} V &= \{a, u_1, \ldots, u_m, v_1, \ldots, v_t, b\} \\ E &= \{(a, u_i) \text{ with capacity } k_i : i \in [m]\} \\ &\quad \cup \{(v_j, b) \text{ with capacity } 1 : j \in [t]\} \\ &\quad \cup \{(u_i, v_j) \text{ with capacity } 1 : |Z_i \cap C_j| \geq 1\} \end{aligned}$$

11: Compute max a-b flow.
12: **if** flow size $< t$ **then return** $\emptyset$ ▷Abort
13: **else** ▷max flow is t
14: $\forall (u_i, v_j)$ with flow add a node in C_j with color i to $\mathcal{S}$
return $\mathcal{S}$

The algorithm constructs a sets $C_1, \ldots, C_t$ such that we will be able to argue that if we can pick a fair set of cluster centers from $C_1 \cup \ldots \cup C_t$ such that *exactly* one point is picked in each C_j then we get a clustering with cluster radius 3γ. Furthermore, if $\gamma \geq r^*_{\text{fair}}$, such a set of centers can be proven to exist. We will then be able to find these centers via a reduction to network flow. The network constructed is the same as in Algorithm 3 although the C_j sets in that algorithm are constructed differently. The only difference is that because we need exactly one point in each of $C_1, C_2, \ldots, C_t$, we need to find a flow of size t rather than a flow of size k. Note that if we are able to construct a flow of $t \leq k$, we can arbitrarily add the cluster centers missing from a class $i \in [m]$ without affecting the clustering radius of the solution.

▶ **Theorem 14.** *If $\gamma \geq r^*_{fair}$ then the above algorithm returns a fair clustering with radius at most 3γ. If $\gamma < r^*_{fair}$ then either the algorithm aborts or it returns a fair clustering with radius at most 3γ.*

Proof. Note that if the algorithm does not abort, the algorithm identifies exactly one point in each of the disjoint sets $C_1, \ldots, C_t$ such that at most k_i points of color i are chosen for each color $i \in [m]$. Since the algorithm did not abort at Step 3 we know that all points in $\mathcal{U}$ are within distance 2γ of some point y_i and hence at most distance $2\gamma + \gamma$ from the selected point in C_i. Hence, we return a fair clustering with covering radius at most 3γ as required. It remains to show that if $\gamma \geq r^*_{\text{fair}}$ then the algorithm does not abort. The algorithm does not abort at line 5 since this would imply there exist $k+1$ points that are $> 2\gamma$ from each other and this implies $r^*_{\text{fair}} > \gamma$. Define $E_j = \{x : d(x, y_j) \leq \gamma\}$ and note that the optimum

solution must pick a point in each E_j since otherwise y_j is not covered within distance γ. Hence, we know it is possible to pick at most k_j points of color j such that exactly one point c_j is picked in each E_j. Note that E_j has a point of color i iff C_j has a point of color i. Hence, it is also possible to pick at most k_i points of color i (for each $i \in [m]$) such that exactly one point c_j is picked in each C_j. Hence, there exists a flow of size t where (u_i, v_j) has flow 1 iff c_j has color i and all edges into b are saturated. ◀

Final algorithm. We now proceed as in the case of FAIR-FLOW (Section 3.2): we can either binary search for the good γ over the continuous range $[d_{\min}, d_{\max}]$ or over the discrete set of all distances between points in $Y \cup D_1 \cup D_2 \cup \ldots \cup D_m$. In the first case, we need $O(\log \log_{1+\epsilon} d_{\max}/d_{\min})$ instantiations of the basic algorithm before we find a clustering with approximation ratio $3(1+\epsilon)$. In the second case, we need to sort $O(k^2m^2)$ distances and then need $O(\log k)$ instantiations.

▶ **Theorem 15.** *There is a 3-approximation for fair k-center clustering with running time $O(kn + m^2k^2 \log k)$.*

Proof. Note that Y and $D_1, D_2, \ldots, D_m$ can be computed in $O(kn)$ time. The flow instance has $O(k)$ nodes and $O(mk)$ edges. Hence, it can be solved in $O(mk^2)$ time [40, 41]. The total running time is therefore $O(kn + m^2k^2 \log k + mk^2 \log k)$ as required. ◀

Enumeration Algorithms for Conjunctive Queries with Projection

Shaleen Deep
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA

Xiao Hu
Department of Computer Sciences, Duke University, Durham, NC, USA

Paraschos Koutris
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA

Abstract

We investigate the enumeration of query results for an important subset of CQs with projections, namely star and path queries. The task is to design data structures and algorithms that allow for efficient enumeration with delay guarantees after a preprocessing phase. Our main contribution is a series of results based on the idea of interleaving precomputed output with further join processing to maintain delay guarantees, which maybe of independent interest. In particular, we design combinatorial algorithms that provide instance-specific delay guarantees in linear preprocessing time. These algorithms improve upon the currently best known results. Further, we show how existing results can be improved upon by using fast matrix multiplication. We also present new results involving tradeoff between preprocessing time and delay guarantees for enumeration of path queries that contain projections. CQs with projection where the join attribute is projected away is equivalent to boolean matrix multiplication. Our results can therefore be also interpreted as sparse, output-sensitive matrix multiplication with delay guarantees.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Query result enumeration, joins, ranking

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.14

Related Version *Full Version*: `https://arxiv.org/pdf/2101.03712.pdf` [10]

Funding This research was supported in part by National Science Foundation grants CRII-1850348 and III-1910014.

Acknowledgements We would like to thank the anonymous reviewers for their careful reading and valuable comments that contributed greatly in improving the manuscript.

1 Introduction

The efficient evaluation of join queries over static databases is a fundamental problem in data management. There has been a long line of research on the design and analysis of algorithms that minimize the total runtime of query execution in terms of the input and output size [33, 21, 20]. However, in many data processing scenarios it is beneficial to split query execution into two phases: the *preprocessing phase*, which computes a space-efficient intermediate data structure, and the *enumeration phase*, which uses the data structure to enumerate the query results as fast as possible, with the goal of minimizing the *delay* between outputting two consecutive tuples in the result. This distinction is beneficial for several reasons. For instance, in many scenarios, the user wants to see one (or a few) results of the query as fast as possible: in this case, we want to minimize the time of the preprocessing phase, such that we can output the first results quickly. On the other hand, a data processing pipeline may require that the result of a query is accessed multiple times by a downstream task: in this case, it is better to spend more time during the preprocessing phase, to guarantee a faster enumeration with smaller delay.

Previous work in the database literature has focused on finding the class of queries that can be computed with $O(|D|)$ preprocessing time (where D is the input database instance) and constant delay during the enumeration phase. The main result in this line of work shows that full (i.e., without projections) acyclic Conjunctive Queries (CQs) admit linear preprocessing time and constant delay [3]. If the CQ is not full but its free variables satisfy the *free-connex* property, the same preprocessing time and delay guarantees can still be achieved. It is also known that for any (possibly non-full) acyclic CQ, it is possible to achieve linear delay after linear preprocessing time [3]. Prior work that uses structural decomposition methods [15] generalized these results to arbitrary CQs with free variables and showed that the projected solutions can be enumerated with $O(|D|^{\texttt{fhw}})$ delay. Moreover, a dichotomy about the classes of conjunctive queries with fixed arities where such answers can be computed with polynomial delay (WPD) is also shown. When the CQ is full but not acyclic, factorized databases uses $O(|D|^{\texttt{fhw}})$ preprocessing time to achieve constant delay, where $\texttt{fhw}$ is the *fractional hypertree width* [14] of the query. We should note here that we can always compute and materialize the result of the query during preprocessing to achieve constant delay enumeration but at the cost of using exponential amount of space in general.

The aforementioned prior work investigates specific points in the preprocessing time-delay tradeoff space. While the story for full acyclic CQs is relatively complete, the same is not true for general CQs, even for acyclic CQs with projections. For instance, consider the simplest such query: $Q_{\texttt{two-path}} = \pi_{x,z}(R(x,y) \bowtie S(y,z))$, which joins two binary relations and then projects out the join attribute. For this query, [3] ruled out a constant delay algorithm with linear time preprocessing unless the boolean matrix multiplication exponent is $\omega = 2$. However, we can obtain $O(|D|)$ delay with $O(|D|)$ preprocessing time. We can also obtain $O(1)$ delay with $O(|D|^2)$ preprocessing by computing and storing the full result. It is worth asking whether there are other interesting points in this tradeoff between preprocessing time and delay. Towards this end, seminal work by Kara et al. [18] showed that for any hierarchical CQ [1] (possibly with projections), there always exists a smooth tradeoff between preprocessing time and delay. This is the first improvement over the results of Bagan et al. [3] in over a decade for queries involving projections. Applied to the query $Q_{\texttt{two-path}}$, the main result of of [18] shows that for any $\epsilon \in [0,1]$, we can obtain $O(|D|^{1-\epsilon})$ delay with $O(|D|^{1+\epsilon})$ preprocessing time.

In this paper, we continue the investigation of the tradeoff between preprocessing time and delay for CQs with projections. We focus on two classes of CQs: *star queries*, which are a popular subset of hierarchical queries, and a useful subset of non-hierarchical queries known as *path queries*. We focus narrowly on these two classes for two reasons. First, star queries are of immense practical interest given their connections to set intersection, set similarity joins and applications to entity matching (we refer the reader to [9] for an overview). The most common star query seen in practice is $Q_{\texttt{two-path}}$. The same holds true for path queries, which are fundamental in graph processing. Second, as we will see in this paper, even for the simple class of star queries, the tradeoff landscape is complex and requires the development of novel techniques. We also present a result on another subset of hierarchical CQs that we call left-deep. Our key insight is to design enumeration algorithms that depend not only on the input size $|D|$, but are also aware of other data-specific parameters such as the output size. To give a flavor of our results, consider the query $Q_{\texttt{two-path}}$, and denote by $\texttt{OUT}_{\bowtie}$ the output of the corresponding query without projections, $R(x,y) \bowtie S(y,z)$. We can show the following result.

[1] Hierarchical CQs are a strict subset of acyclic CQs.

Queries	Preprocessing	Delay	Source
Arbitrary acyclic CQ	$O(\|D\|)$	$O(\|D\|)$	[3]
Free-connex CQ (projections)	$O(\|D\|)$	$O(1)$	[3]
Full CQ	$O(\|D\|^{\texttt{fhw}})$	$O(1)$	[22]
Full CQ	$O(\|D\|^{\texttt{subw}} \log \|D\|)$	$O(1)$	[1]
Hierarchical CQ (with projections)	$O(\|D\|^{1+(\texttt{w}-1)\epsilon})$	$O(\|D\|^{1-\epsilon})$ $\epsilon \in [0,1]$	[18]
Star query with k relations (with projections)	$O(\|D\|)$	$O(\frac{\|D\|^{k/(k-1)}}{\|\texttt{OUT}_{\bowtie}\|^{1/(k-1)}})$	this paper
Path query with k relations (with projections)	$O(\|D\|^{2-\epsilon/(k-1)})$	$O(\|D\|^{\epsilon})$, $\epsilon \in [0,1)$	this paper
Left-deep hierarchical CQ (with projections)	$O(\|D\|)$	$O(\|D\|^{k}/\|\texttt{OUT}_{\bowtie}\|)$	this paper
Two path query (with projections)	$O(\|D\|^{\omega\cdot\epsilon})$	$O(\|D\|^{1-\epsilon})$, $\epsilon \in [\frac{2}{\omega+1}, 1]$	this paper

Figure 1 Preprocessing time and delay guarantees for different queries. $|\texttt{OUT}_{\bowtie}|$ denotes the size of join query under consideration but without any projections. $\texttt{subw}$ denotes the submodular width of the query. For each class of query, the total running time is $O(\min\{|D|\cdot|\texttt{OUT}_{\pi}|, |D|^{\texttt{subw}} \log |D| + |\texttt{OUT}_{\pi}|\})$ where $|\texttt{OUT}_{\pi}|$ denotes the size of the query result. See the related work section for more discussion on best running times for two path and star queries.

▶ **Theorem 1.** *Given a database instance D, we can enumerate the output of $Q_{\textit{two-path}} = \pi_{x,z}(R(x,y) \bowtie S(y,z))$ with preprocessing time $O(|D|)$ and delay $O(|D|^2/|\texttt{OUT}_{\bowtie}|)$.*

At this point, the reader may wonder about the improvement obtained from the above result. [18] implies that with preprocessing time $O(|D|)$, the delay guarantee in the worst-case is $O(|D|)$. This raises the question whether the delay from Theorem 1 is truly an algorithmic improvement rather than an improved analysis of [18]. We answer the question positively. Specifically, we show that there exists a database instance where the delay obtained from Theorem 1 is a polynomial improvement over the actual guarantee [18] and not just the worst-case. When the preprocessing time is linear, the delay implied by our result is dependent on the size of the full join. In the worst case where $|\texttt{OUT}_{\bowtie}| = \Theta(|D|^2)$, we actually obtain the best delay, which will be constant. Compare this to the result of [18], which would require nearly $O(|D|^2)$ preprocessing time to achieve the same guarantee. On the other hand, if $|\texttt{OUT}_{\bowtie}| = \Theta(|D|)$, we obtain only a linear delay guarantee of $O(|D|)^2$. The reader may wonder how our result compares in general with the tradeoff in [18] in the worst-case; we will show that we can always get at least as good of a tradeoff point as the one in [18]. Figure 1 summarizes the prior work and the results present in this paper.

Our Contribution. In this paper, we improve the state-of-the-art on the preprocessing time-delay tradeoff for a subset of CQs with projections. We summarize our main technical contributions below (highlighted in Figure 1):

1. Our main contribution consists of a novel algorithm (Theorem 7 in Section 4) that achieves output-dependent delay guarantees for star queries after linear preprocessing

[2] We do not need to consider the case where $|\texttt{OUT}_{\bowtie}| \leq |D|$, since then we can simply materialize the full result during the preprocessing time using constant delay enumeration for queries without projections [23].

time. Specifically, we show that for the query $\pi_{x_1,\ldots,x_k}(R_1(x_1,y) \bowtie \cdots \bowtie R_k(x_k,y))$ we can achieve delay $O(|D|^{k/(k-1)}/|\texttt{OUT}_\bowtie|^{1/k-1})$ with linear preprocessing. Our key idea is to identify an appropriate degree threshold to split a relation into partitions of *heavy* and *light*, which allows us to perform efficient enumeration. For star queries, our result implies that there exists no smooth tradeoff between preprocessing time and delay guarantees as stated in [18] for the class of hierarchical queries.

2. We introduce the novel idea of *interleaving* join query computation in the context of enumeration algorithms which forms the foundation for our algorithms, and may be of independent interest. Specifically, we show that it is possible to union the output of two algorithms $\mathcal{A}$ and $\mathcal{A}'$ with δ delay guarantee where $\mathcal{A}$ enumerates query results with δ delay guarantees but $\mathcal{A}'$ does not. This technique allows us to compute a subset of a query *on-the-fly* when enumeration with good delay guarantees is impossible (Lemma 4 and Lemma 5) in Section 3.
3. We show how fast matrix multiplication can be used to obtain a tradeoff between preprocessing time and delay that further improves upon the tradeoff in [18]. We also present an algorithm for left-deep hierarchical queries with linear preprocessing time and output-dependent delay guarantees (Section 5).
4. Finally, we present new results on preprocessing time-delay tradeoffs for a non-hierarchical query with projections, for the class of path queries (Section 6). A path query has the form $\pi_{x_1,x_{k+1}}(R_1(x_1,x_2) \bowtie \cdots \bowtie R_k(x_k,x_{k+1}))$. Our results show that we can achieve delay $O(|D|^\epsilon)$ with preprocessing time $O(|D|^{2-\epsilon/(k-1)})$ for any $\epsilon \in [0,1)$.

2 Problem Setting

In this section, we present the basic notation and terminology.

2.1 Conjunctive Queries

In this paper, we will focus on the class of *conjunctive queries* (CQs), which we denote as

$$Q = \pi_{\mathbf{y}}(R_1(\mathbf{x}_1) \bowtie R_2(\mathbf{x}_2) \bowtie \ldots \bowtie R_n(\mathbf{x}_n))$$

Here, the symbols $\mathbf{y}, \mathbf{x}_1, \ldots, \mathbf{x}_n$ are vectors that contain *variables* or *constants*. We say that Q is *full* if there is no projection. We will typically use the symbols $x, y, z, \ldots$ to denote variables, and $a, b, c, \ldots$ to denote constants. We use $Q(D)$ to denote the result of the query Q over input database D.

In this paper, we will focus on CQs that have no constants and no repeated variables in the same atom (both cases can be handled within a linear time preprocessing step, so this assumption is without any loss of generality). Such a query can be represented equivalently as a *hypergraph* $\mathcal{H}_Q = (\mathcal{V}_Q, \mathcal{E}_Q)$, where $\mathcal{V}_Q$ is the set of variables, and for each hyperedge $F \in \mathcal{E}_Q$ there exists a relation R_F with variables F.

We will be particularly interested in two families of CQs that are fundamental in query processing, star and path queries. The *star query* with k relations is expressed as:

$$Q_k^* = R_1(\mathbf{x}_1, \mathbf{y}) \bowtie R_2(\mathbf{x}_2, \mathbf{y}) \bowtie \cdots \bowtie R_k(\mathbf{x}_k, \mathbf{y})$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_k$ have disjoint sets of variables. The *path query* with k (binary) relations is expressed as:

$$P_k = R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie \cdots \bowtie R_k(x_k, x_{k+1})$$

In Q_k^*, variables in each relation R_i are partitioned into two sets: variables $\mathbf{x}_i$ that are present only in R_i and a common set of join variables $\mathbf{y}$ present in every relation.

Hierarchical Queries. A CQ Q is *hierarchical* if for any two of its variables, either the sets of atoms in which they occur are disjoint or one is contained in the other [29]. For example, Q_k^* is hierarchical for any k, while P_k is hierarchical only when $k \leq 2$.

Join Size Bounds. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph, and $S \subseteq \mathcal{V}$. A weight assignment $\mathbf{u} = (u_F)_{F \in \mathcal{E}}$ is called a *fractional edge cover* of S if (i) for every $F \in \mathcal{E}, u_F \geq 0$ and (ii) for every $x \in S, \sum_{F: x \in F} u_F \geq 1$. The *fractional edge cover number* of S, denoted by $\rho^*_{\mathcal{H}}(S)$ is the minimum of $\sum_{F \in \mathcal{E}} u_F$ over all fractional edge covers of S. We write $\rho^*(\mathcal{H}) = \rho^*_{\mathcal{H}}(\mathcal{V})$.

Tree Decompositions. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph of a CQ Q. A *tree decomposition* of $\mathcal{H}$ is a tuple $(\mathcal{T}, (\mathcal{B}_t)_{t \in V(\mathcal{T})})$ where $\mathcal{T}$ is a tree, and every $\mathcal{B}_t$ is a subset of $\mathcal{V}$, called the *bag* of t, such that

1. each edge in $\mathcal{E}$ is contained in some bag; and
2. for each variable $x \in \mathcal{V}$, the set of nodes $\{t \mid x \in \mathcal{B}_t\}$ form a connected subtree of $\mathcal{T}$.

The *fractional hypertree width* of a decomposition is defined as $\max_{t \in V(\mathcal{T})} \rho^*(\mathcal{B}_t)$, where $\rho^*(\mathcal{B}_t)$ is the minimum fractional edge cover of the vertices in $\mathcal{B}_t$. The fractional hypertree width of a query Q, denoted $\texttt{fhw}(Q)$, is the minimum fractional hypertree width among all tree decompositions of its hypergraph. We say that a query is *acyclic* if $\texttt{fhw}(Q) = 1$.

Computational Model. To measure the running time of our algorithms, we will use the uniform-cost RAM model [16], where data values as well as pointers to databases are of constant size. Throughout the paper, all complexity results are with respect to data complexity, where the query is assumed fixed.

2.2 Fast Matrix Multiplication

Let A be a $U_1 \times U_3$ matrix and C be a $U_3 \times U_2$ matrix over any field $\mathcal{F}$. $A_{i,j}$ is the shorthand notation for entry of A located in row i and column j. The matrix product is given by $(AC)_{i,j} = \sum_{k=1}^{U_3} A_{i,k} C_{k,j}$. Algorithms for fast matrix multiplication are of extreme theoretical interest given its fundamental importance. We will frequently use the following folklore lemma about rectangular matrix multiplication.

▶ **Lemma 2.** *Let ω be the smallest constant such that an algorithm to multiply two $n \times n$ matrices that runs in time $O(n^\omega)$ is known. Let $\beta = \min\{U, V, W\}$. Then fast matrix multiplication of matrices of size $U \times V$ and $V \times W$ can be done in time $O(UVW\beta^{\omega-3})$.*

Observe that in Lemma 2, matrix multiplication cost dominates the time required to construct the input matrices (if they have not been constructed already) for all $\omega \geq 2$. Fixing $\omega = 2$, rectangular matrix multiplication can be done in time $O(UVW/\beta)$. A long line of research on fast square matrix multiplication has dropped the complexity to $O(n^\omega)$, where $2 \leq \omega < 3$. The current best known value is $\omega = 2.3729$ [13], but it is believed that the actual value is 2.

2.3 Problem Statement

Given a Conjunctive Query Q and an input database D, we want to enumerate the tuples in $Q(D)$ in any order. We will study this problem in the enumeration framework similar to that of [27], where an algorithm can be decomposed into two phases:

- **Preprocessing phase:** it computes a data structure that takes space S_p in *preprocessing time* T_p.
- **Enumeration phase:** it outputs $Q(D)$ with no repetitions. This phase has access to any data structures constructed in the preprocessing phase and can also use additional space of size S_e. The *delay* δ is defined as the maximum time duration between outputting any pair of consecutive tuples (and also the time to output the first tuple, and the time to notify that the enumeration phase has completed).

In this work, our goal is to study the relationship between the preprocessing time T_p and delay δ for a given CQ Q. Ideally, we would like to achieve the best possible delay in linear preprocessing time. As Figure 1 shows, when Q is full, with $T_p = O(|D|^{\texttt{fhw}})$, we can enumerate the results with constant delay $O(1)$ [22]. In the particular case where Q is acyclic i.e. $\texttt{fhw} = 1$, we can achieve constant delay with only linear preprocessing time. On the other hand, [3] shows that for every acyclic CQ, we can achieve linear delay $O(|D|)$ with linear preprocessing time $O(|D|)$.

Recently, [18] showed that it is possible to get a tradeoff between the two extremes, for the class of hierarchical queries. Note that hierarchical queries are acyclic but not necessarily free-connex. This is the first non-trivial result that improves upon the linear delay guarantees given by [3] for queries with projections.

▶ **Theorem 3** (due to [18]). *Consider a hierarchical CQ Q with factorization width w, and an input instance D. Then, for any $\epsilon \in [0, 1]$ there exists an algorithm that can preprocess D in time $T_p = O(|D|^{1+(\mathsf{w}-1)\epsilon})$ and space $S_p = O(|D|^{1+(\mathsf{w}-1)\epsilon})$ such that we can enumerate the query output with*

$$\textit{delay } \delta = O(|D|^{1-\epsilon}) \qquad \textit{space } S_e = O(1).$$

The factorization width w of a query, originally introduced as $s^{\uparrow}$ [23], is a generalization of the fractional hypertree width from boolean to arbitrary CQs. For $\pi_{\mathbf{x}_1,\ldots,\mathbf{x}_k}(Q_k^*)$, the factorization width is $\mathsf{w} = k$. Observe that preprocessing time T_p is always smaller than the time required to evaluate the full join result. This is because if $T_p = \Theta(|\texttt{OUT}_{\bowtie}|)$, we can evaluate the full join and deduplicate the projection output, allowing us to obtain constant delay in the enumeration phase. This implies that ϵ can only take values between 0 and $(\log_{|D|} |\texttt{OUT}_{\bowtie}| - 1)/(\mathsf{w} - 1)$.

3 Helper Lemmas

Before we present the proof of our main results, we discuss three useful lemmas which will be used frequently, and may be of independent interest for enumeration algorithms. The first two lemmas are based on the key idea of *interleaving query results* which we describe next. We note that idea of interleaving computation has been explored in the past to develop dynamic algorithms with good worst-case bounds using static data structures [24].

We say that an algorithm $\mathcal{A}$ provides no delay guarantees to mean that its delay guarantee is its total execution time. In other words, if an algorithm requires time T to complete, its delay guarantee is upper bounded by T. Since we are using the uniform-cost RAM model, each operation takes one unit of time.

▶ **Lemma 4.** *Consider two algorithms $\mathcal{A}$ and $\mathcal{A}'$ such that*
1. *$\mathcal{A}$ enumerates query results in total time at most T with no delay guarantees.*
2. *$\mathcal{A}'$ enumerates query results with delay δ and runs in total time at least T'.*
3. *The outputs of $\mathcal{A}$ and $\mathcal{A}'$ are disjoint.*
4. *T and T' are provided as input to the algorithm.*

Then, the union of the outputs of $\mathcal{A}$ and $\mathcal{A}'$ can be enumerated with delay $c \cdot \delta \cdot \max\{1, T/T'\}$ for some constant c.

Lemma 4 tells us that as long as $T = O(T')$, the output of $\mathcal{A}$ and $\mathcal{A}'$ can be combined without giving up on delay guarantees by pacing the output of $\mathcal{A}'$. Note that we need to know the exact values of T and T' (by calculating the number of operations in the algorithms $\mathcal{A}$ and $\mathcal{A}'$ to bound the running time). The next lemma introduces our second key idea of interleaving stored output result with *on-the-fly* query computation (the full algorithm and proof can be found in [10]).

▶ **Lemma 5.** *Consider an algorithm $\mathcal{A}$ that enumerates query results in total time at most T with no delay guarantees, where T is known in advance. Suppose that J output tuples have been stored apriori with no duplicate tuples, where $J \leq T$. Then, there exists an algorithm that enumerates the output with delay guarantee $\delta = O(T/J)$.*

The final helping lemma allows us to enumerate the union of (possibly overlapping) results of m different algorithms where each algorithm outputs its result according to a total order $\preceq$, such that the union is also enumerated in sorted order according to $\preceq$. This lemma is based on the idea presented as Fact 3.1.4 in [19].

▶ **Lemma 6.** *Consider m algorithms $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_m$ such that each $\mathcal{A}_i$ enumerates its output L_i with delay $O(\delta)$ according to the total order $\preceq$. Then, the union of their output can be enumerated (without duplicates) with $O(m \cdot \delta)$ delay and in sorted order according to $\preceq$.*

Directly implied by Lemma 6 is the fact that the *list merge* problem can be enumerated with delay guarantees: Given m lists $L_1, L_2, \cdots, L_m$ whose elements are drawn from a common domain, if elements in L_i are distinct (i.e no duplicates) and ordered according to $\preceq$, then the union of all lists $\bigcup_{i=1}^{m} L_i$ can be enumerated in sorted order given by $\preceq$ with delay $O(m)$. Note that the enumeration algorithm $\mathcal{A}_i$ degenerates to going over elements one by one in list L_i, which has $O(1)$ delay guarantee as long as indexes/pointers within L_i are well-built. Throughout the paper, we use this primitive as $\textsc{ListMerge}(L_1, L_2, \cdots, L_m)$.

4 Star Queries

In this section, we study enumeration algorithms for the star query $\pi_r(Q_k^*)$ where $r \subseteq \bigcup_{i \in \{1,2,\cdots,k\}} \mathbf{x}_i$. Our main result is Theorem 7 that we present below. We first present a detailed discussion on how our result is an improvement over prior work in Subsection 4.1. Then, we present a warm-up proof for $\pi_r(Q_k^*)$ in Subsection 4.2, followed by the proof for the general result in Subsection 4.3.

▶ **Theorem 7.** *Consider the star query[3] with projection $\pi_r(Q_k^*)$ where $r \subseteq \bigcup_{i \in \{1,2,\cdots,k\}} \mathbf{x}_i$ and an instance D. There exists an algorithm with preprocessing time $T_p = O(|D|)$ and preprocessing space $S_p = O(|D|)$, such that we can enumerate $Q_k^*(D)$ with*

$$\text{delay } \delta = O\left(\frac{|D|^{k/k-1}}{|OUT_{\bowtie}|^{1/k-1}}\right) \text{ and space } S_e = O(|D|).$$

[3] We assume that r contains at least one variable from each $\mathbf{x}_i$. Otherwise, we can remove relations with no projection variables after the preprocessing phase.

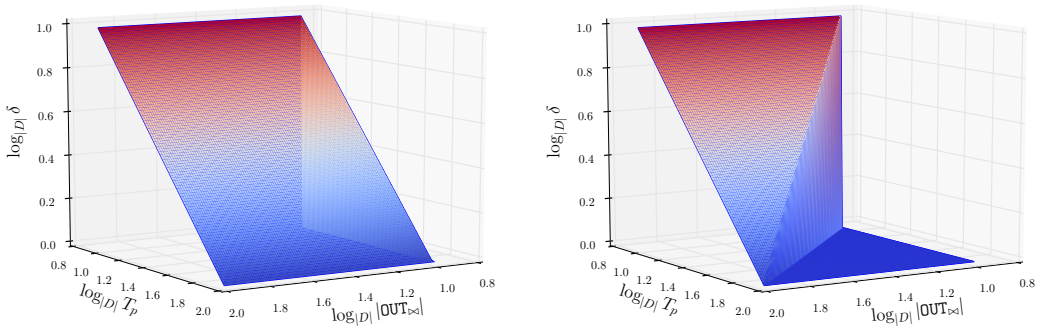


Figure 2 Worst-case tradeoffs given by Theorem 3 without (left) and with (right) taking $|\mathtt{OUT}_{\bowtie}|$ into consideration.

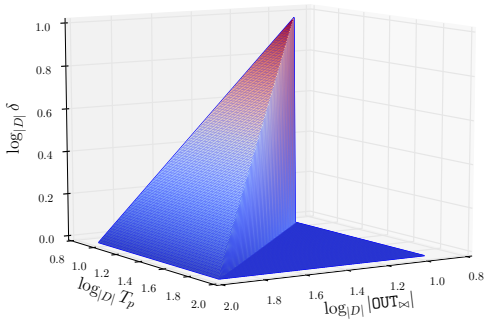


Figure 3 Theorem 7 for $k = 2$.

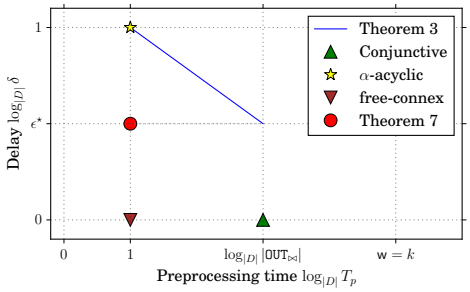


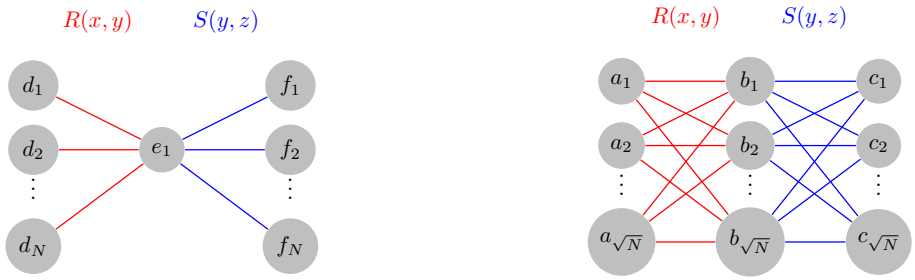
Figure 4 Trade-off in the worst-case for star query.

In the above theorem, the delay depends on the full join result size $|\mathtt{OUT}_{\bowtie}| = |Q_k^*(D)|$. As the join size increases, the algorithm can obtain better delay guarantees. In the extreme case when $|\mathtt{OUT}_{\bowtie}| = \Theta(|D|^k)$, it achieves constant delay with linear time preprocessing. In the other extreme, when $|\mathtt{OUT}_{\bowtie}| = \Theta(|D|)$, it achieves linear delay.

When $|\mathtt{OUT}_{\bowtie}|$ has linear size, we can compute and materialize the result of the query in linear preprocessing time and achieve constant delay enumeration. Generalizing this observation, when T_p is sufficient to evaluate the full join result, we can always achieve constant delay.

4.1 Comparison with Prior Work

It is instructive now to compare the worst-case delay guarantee obtained by Theorem 3 for $Q_k^*(D)$ with Theorem 7. Suppose that we want to achieve delay $\delta = O(|D|^{1-\epsilon})$ for some $\epsilon \in [0, (\log_{|D|}|\mathtt{OUT}_{\bowtie}| - 1)/(k-1)]$. Theorem 3 tells us that this requires $O(|D|^{1+\epsilon(k-1)})$ preprocessing time. Then, it holds that:



(a) Database D_0 with full join size N^2.

(b) Database D_1 with full join size $N^{3/2}$.

Figure 5 $D_0 \cup D_1$ forms a database where Theorem 7 improves the delay of Theorem 3.

$$|D|^{1-\epsilon} \geq |D|^{1-\frac{(\log_{|D|}|\texttt{OUT}_\bowtie|-1)}{k-1}} = |D|^{\frac{k-\log_{|D|}|\texttt{OUT}_\bowtie|}{k-1}} = |D|^{k/k-1}/|\texttt{OUT}_\bowtie|^{1/k-1}$$

In other words, either we have enough preprocessing time to materialize the output and achieve constant delay, or we can achieve the desirable delay with linear preprocessing time.

Figure 2, Figure 3 and Figure 4 show the existing and new tradeoff results. Figure 2 shows the tradeoff curve obtained from Theorem 3 by adding $|\texttt{OUT}_\bowtie|$ as a third dimension, and adding the optimization for constant delay when $T_p \geq O(|\texttt{OUT}_\bowtie|)$. Figure 3 shows the tradeoff obtained from our result, while Figure 4 shows other existing results for a fixed value of $|\texttt{OUT}_\bowtie|$. For a fixed value of $|\texttt{OUT}_\bowtie|$, the delay guarantee does not change in Figure 3 as we increase T_p from $|D|$ to $|\texttt{OUT}_\bowtie|$. It remains an open question to further decrease the delay if we allow more preprocessing time. Such an algorithm would correspond to a curve connecting the red point(●) and the green triangle(▲) in Figure 4.

Our results thus imply that, depending on $|\texttt{OUT}_\bowtie|$, one must choose a different algorithm to achieve the optimal tradeoff between preprocessing time and delay. Since $|\texttt{OUT}_\bowtie|$ can be computed in linear time (using a simple adaptation of Yannakakis algorithm [33, 25]), this can be done without affecting the preprocessing bounds.

Next, we show how our result provides an algorithmic improvement over Theorem 3. Consider the instances D_0, D_1 depicted in Figure 5a and Figure 5b respectively, and assume we want to use linear preprocessing time. For D_1, the algorithm of Theorem 3 materializes nothing, since no y valuation has a degree of $O(|D|^0)$, and the delay will be $\Theta(\sqrt{N})$. No materialization also occurs for D_0, but here the delay will be $O(1)$. It is easy to check that our algorithm matches the delay on both instances. Now, consider the instance $D = D_0 \cup D_1$. The input size for D is $\Theta(N)$, while the full join size is $N^{3/2} + N^2 = \Theta(N^2)$. The algorithm of Theorem 3 will again achieve only a $\Theta(\sqrt{N})$ delay, since after the linear time preprocessing no y valuations can be materialized. In contrast, our algorithm still guarantees a constant delay. This algorithmic improvement is a result of the careful overlapping of the constant-delay computation for instance D_0 with the computation for D_1.

The above construction can be generalized as follows. Let $\alpha \in (0, 1)$ be some constant. D_0 remains the same. For D_1, we construct R to be the cross product of N^α x-values and $N^{1-\alpha}$ y-values, and S to be the cross product of N^α z-values and $N^{1-\alpha}$ y-values. As before, let $D = D_0 \cup D_1$. The input size for D is $\Theta(N)$, while the full join size is $N^{2-\alpha} + N^2 = \Theta(N^2)$. Hence, our algorithm achieves constant delay with linear preprocessing time. In contrast, the algorithm of Theorem 3 achieves $\Theta(N^{1-\alpha})$ delay with linear preprocessing time. In fact, the $\Theta(N^{1-\alpha})$ delay occurs even if we allow $O(N^{1+\epsilon})$ preprocessing time for any $\epsilon < \alpha$. We can now use the same idea to show that there also exists an instance where achieving constant delay using Theorem 3 requires near quadratic preprocessing time (see the Appendix in [10]).

In the rest of the paper, for simplicity of exposition, we assume that all variable vectors $\mathbf{x}_i, \mathbf{y}$ in Q_k^* are singletons (i.e, all the relations are binary) and $r = \{x_1, x_2, \ldots, x_k\}$. The proof for the general query is a straightforward extension of the binary case.

4.2 Warm-up: Two-Path Query

As a warm-up step, we will present an algorithm for the query $Q_{\texttt{two-path}} = \pi_{x,z}(R(x,y) \bowtie S(y,z))$ that achieves $O(|D|^2/|\texttt{OUT}_\bowtie|)$ delay with linear preprocessing time.

At a high level, we will decompose the join into two subqueries with disjoint outputs. The subqueries will be generated based on whether a valuation for x is *light* or not based on its degree in relation R. For all light valuations of x (degree at most δ), we will show that their enumeration is achievable with delay δ. For the heavy x valuations, we will show that they also can be computed *on-the-fly* while maintaining the delay guarantees.

Preprocessing Phase. We first process the input relations such that we remove any dangling tuples. During the preprocessing phase, we will store the input relations as a hash map and sort the valuations for x in increasing order of their degree. Using any comparison based sorting technique requires $\Omega(|D| \log |D|)$ time in general. Thus, if we wish to remove the $\log |D|$ factor, we must use non-comparison based sorting algorithms. In this paper, we will use count sort [8] which has complexity $O(|D| + r)$ where r is the range of the non-negative key values. However, we need to ensure that all relations in the database D satisfy the bounded range requirement. This can be easily accomplished by introducing a bijective function $f : \mathbf{dom}(D) \to \{1, 2, \ldots, |D|\}$ that maps all values in the active domain of the database to some integer between 1 and $|D|$ (both inclusive). Both f and its inverse f^{-1} can be stored as hash tables as follows: suppose there is a counter $c \leftarrow 1$. We perform a linear pass over the database and check if some value $v \in \mathbf{dom}(D)$ has been mapped or not (by checking if there exists an entry $f(v)$). If not, we set $f(v) = c, f^{-1}(c) = v$ and increment c. Once the hash tables f and f^{-1} have been created, we modify the input relation R (and S similarly) by replacing every tuple $t \in R$ with tuple $t' = f(t)$. Since the mapping is a relabeling scheme, such a transformation preserves the degree of all the values. The codomain of f is also equipped with a total order $\preceq$ (we will use $\leq$). Note that f is not an order-preserving transformation in general but this property is not required in any of our algorithms.

Next, for every tuple $t \in R(x,y)$, we create a hash map with key $\pi_x(t)$ and the value is a list $\pi_y(t)$; and for every tuple $t \in S(y,z)$, we create a hash map with key $\pi_y(t)$ and the value is a list $\pi_z(t)$. For the second hash map, we sort the value list using sort order $\preceq$ for each key, once each tuple $t \in S(y,z)$ has been processed. Finally, we sort all values in $\pi_x(R)$ in increasing order of their degree in R (i.e $|\sigma_{x=v_i} R(x,y)|$ is the sort key). Let $\mathcal{L} = \{v_1, \ldots, v_n\}$ denote the ordered set of these values sorted by their degree and let $d_1, \ldots, d_n$ be their respective degrees. Creating the sorted list $\mathcal{L}$ takes $O(|D|)$ time since the degrees d_i satisfy the bounded range requirement (i.e $1 \leq d_i \leq |D|$). Next, we identify the smallest index i^* such that

$$\sum_{v:\{v_1, v_2, \ldots, v_{i^*}\}} |R(v,y) \bowtie S(y,z)| \geq \sum_{v:\{v_{i^*+1}, \ldots, v_n\}} |R(v,y) \bowtie S(y,z)| \tag{1}$$

This can be computed by doing a linear pass on $\mathcal{L}$ using a simple adaptation of Yannakakis algorithm [33, 25]. This entire phase takes time $O(|D|)$.

Enumeration Phase. The enumeration algorithm interleaves the following two loops using the construction in Lemma 4. Specifically, it will spend an equal amount of time (a constant) before switching to the computation of the other loop.

Algorithm 1 EnumTwoPath.

for $i = 1, \ldots, i^*$ **do** — run for $O(1)$ time then switch
  Let $\pi_y(\sigma_{x=v_i}(R)) = \{u_1, u_2, \cdots, u_\ell\}$;
  output $(v_i, f^{-1}(\text{ListMerge}(\pi_z\sigma_{y=u_1}S, \pi_z\sigma_{y=u_2}S, \cdots, \pi_z\sigma_{y=u_\ell}S)))$[4]
for $i = i^* + 1, \ldots, n$ **do** — run for $O(1)$ time then switch
  Let $\pi_y(\sigma_{x=v_i}(R)) = \{u_1, u_2, \cdots, u_\ell\}$;
  output $(v_i, f^{-1}(\text{ListMerge}(\pi_z\sigma_{y=u_1}S, \pi_z\sigma_{y=u_2}S, \cdots, \pi_z\sigma_{y=u_\ell}S)))$

The algorithm alternates between low-degree and high-degree values in $\mathcal{L}$. The main idea is that, for a given $v_i \in \mathcal{L}$, we can enumerate the result of the subquery $\sigma_{x=v_i}(Q_{\texttt{two-path}})$ with delay $O(d_i)$. This can be accomplished by observing that the subquery is equivalent to list merging and so we can use Lemma 6.

▶ **Lemma 8.** *For the query $Q_{\texttt{two-path}}$ and an instance D, we can enumerate $Q_{\texttt{two-path}}(D)$ with delay $\delta = O(|D|^2/|\texttt{OUT}_\bowtie|)$ and $S_e = O(|D|)$.*

The reader should note that the delay of $\delta = O(|D|^2/|\texttt{OUT}_\bowtie|)$ is only an upper bound. Depending on the skew present in the database instance, it is possible that Algorithm 1 achieves much better delay guarantees in practice (as shown in the full version).

4.3 Proof of Main Theorem

We now generalize Algorithm 1 for any star query. At a high level, we will decompose the join query $\pi_{x_1,\ldots,x_k}(Q_k^*)$ into a union of $k+1$ subqueries whose output is a partition of the result of original query. These subqueries will be generated based on whether a value for some x_i is *light* or not. We will show if any of the values for x_i is light, the enumeration delay is small. The $(k+1)$-th subquery will contain heavy values for all attributes. Our key idea again is to interleave the join computation of the *heavy* subquery with the remaining light subqueries.

Preprocessing Phase. Assume all relations are reduced without dangling tuples, which can be achieved in linear time [33]. The full join size $|\texttt{OUT}_\bowtie|$ can also be computed in linear time. Similar to the preprocessing phase in the previous section, we construct the hash tables f, f^{-1} to perform the domain compression and modify all the input relations by replacing tuple t with $f(t)$. Set $\Delta = (2 \cdot |D|^k/|\texttt{OUT}_\bowtie|)^{\frac{1}{k-1}}$. For each relation R_i, a value v for attribute x_i is *heavy* if its degree (i.e $|\pi_y\sigma_{x_i=v}R(x_i, y)|$) is greater than Δ, and *light* otherwise. Moreover, a tuple $t \in R_i$ is identified as heavy or light depending on whether $\pi_{x_i}(t)$ is heavy or light. In this way, each relation R is divided into two relations R^h and R^ℓ, containing heavy and light tuples respectively in time $O(|D|)$. The original query can be decomposed into subqueries of the following form:

$$\pi_{x_1,x_2,\cdots,x_k}(R_1^? \bowtie R_2^? \bowtie \cdots \bowtie R_k^?)$$

[4] Abusing the notation, $f^{-1}(\mathcal{B})$ for some ordered list (or tuple) $\mathcal{B}$ returns an ordered list (tuple) $\mathcal{B}'$ where $\mathcal{B}'(i) = f^{-1}(\mathcal{B}(i))$.

where ? can be either h, ℓ or $\star$. Here, $R_i^\star$ simply denotes the original relation R_i. However, care must be taken to generate the subqueries in a way so that there is no overlap between the output of any subquery. In order to do so, we create k subqueries of the form

$$Q_i = \pi_{x_1,\ldots,x_k}(R_1^h \bowtie \cdots \bowtie R_{i-1}^h \bowtie R_i^\ell \bowtie R_{i+1}^\star \bowtie \cdots \bowtie R_k^\star)$$

In subquery Q_i, relation R_i has superscript ℓ, all relations $R_1, \ldots, R_{i-1}$ have superscript h and relations $R_{i+1}, \ldots, R_k$ have superscript $\star$. The $(k+1)$-th query with all ? as h is denoted by Q_H. Note that each output tuple t is generated by exactly one of the Q_i and thus the output of all subqueries is disjoint. This implies that each $f^{-1}(t)$ is also generated by exactly one subquery. Similar to the preprocessing phase of two path query, we store all R_i^ℓ and R_i^h in hashmaps where the values in the maps are lists sorted in lexicographic order.

Enumeration Phase. We next describe how enumeration is performed. The key idea is the following: We will show that for $Q_L = Q_1 \cup \cdots \cup Q_k$, we can enumerate the result in delay $O(\Delta)$. Since Q_H contains all heavy valuations from all relations, we compute its join *on-the-fly* by alternating between some subquery in Q_L and Q_H. This will ensure that we can give some output to the user with delay guarantees and also make progress on computing the full join of Q_H. Our goal is to reason about the running time of enumerating Q_L (denoted by T_L) and the running time of Q_H (denoted by T_H) and make sure that while we compute Q_H, we do not run out of the output from Q_L.

Next, we introduce the algorithm that enumerates output for any specific valuation v of attribute x_i, which is described in Lemma 9. This algorithm can be viewed as another instantiation of Lemma 6.

▶ **Lemma 9.** *Consider an arbitrary value $v \in \mathbf{dom}(x_i)$ with degree d in relation $R_i(x_i, y)$. Then, its query result $\pi_{x_1,x_2,\cdots,x_k}\sigma_{x_i=v}R_1^h(x_1, y) \bowtie R_2^h(x_2, y) \bowtie \cdots \bowtie R_i(x_i, y) \bowtie \cdots \bowtie R_k^\star(x_k, y)$ can be enumerated with $O(d)$ delay guarantee.*

Let $c^\star$ be an upper bound on the number of operations in each iteration of ListMerge. This can be calculated by counting the number of operations in the exact implementation of the algorithm. Directly implied by Lemma 9, the result of any subquery in Q_L can be enumerated with delay $O(\Delta)$. Let Q_H^* denote the corresponding full query of Q_H, i.e, the head of Q_H^* also includes the variable y (Q_L^* is defined similarly). Then, Q_H^* can be evaluated in time $T_H \leq c^\star \cdot |Q_H^*| \leq c^\star \cdot |\mathtt{OUT}_\bowtie|/2$ by using ListMerge on subquery Q_H. This follows from the bound $|Q_H^*| \leq |D| \cdot (|D|/\Delta)^{k-1}$ and our choice of $\Delta = (2 \cdot |D|^k/|\mathtt{OUT}_\bowtie|)^{\frac{1}{k-1}}$. Since $|Q_H^*| + |Q_L^*| = |\mathtt{OUT}_\bowtie|$, it holds that $|Q_L^*| \geq |\mathtt{OUT}_\bowtie|/2$ given our choice of Δ. Also, the running time T_L is lower bounded by $|Q_L^*|$ (since we need at least one operation for every result). Thus, $T_L \geq |\mathtt{OUT}_\bowtie|/2$.

We are now ready to apply Lemma 4 with the following parameters:

1. $\mathcal{A}$ is the full join computation of Q_H and $T = c^\star \cdot |\mathtt{OUT}_\bowtie|/2$.
2. $\mathcal{A}'$ is the enumeration algorithm applied to Q_L with delay guarantee $\delta = O(\Delta)$ and $T' = |\mathtt{OUT}_\bowtie|/2$.
3. T and T' are fixed once $|\mathtt{OUT}_\bowtie|, \Delta$ and the constant $c^\star$ are known.

By construction, the outputs of Q_H and Q_L are also disjoint. Thus, the conditions of Lemma 4 apply and we obtain a delay of $O(\Delta)$.

4.4 Interleaving with Join Computation

Theorem 7 obtains poor delay guarantees when the full join size $|\mathtt{OUT}_\bowtie|$ is close to input size $|D|$. In this section, we present an alternate algorithm that provides good delay guarantees in this case. The algorithm is an instantiation of Lemma 5 on the star query, which

degenerates to computing as many distinct output results as possible in limited preprocessing time. An observation is that for each valuation u of attribute y, the cartesian product $\times_{i\in\{1,2,\cdots,k\}}\pi_{x_i}\sigma_{y=u}R_i(x_i,y)$ is a subset of output results without duplication. Thus, this subset of output result is readily available since no deduplication needs to be performed. Similarly, after all relations are reduced, it is also guaranteed that each valuation of attribute x_i of relation R_i generates at least one output result. Thus, $\max_{i=1}^{k}|\mathbf{dom}(x_i)|$ results are also readily available that do not require deduplication. We define J as the larger of the two quantities, i.e, $J = \max\left\{\max_{i=1}^{k}|\mathbf{dom}(x_i)|, \max_{u\in\mathbf{dom}(y)}\prod_{i=1}^{k}|\sigma_{y=u}R_i(x_i,y)|\right\}$. Together with these observations, we can achieve the following theorem.

▶ **Theorem 10.** *Consider star query $\pi_{x_1,\ldots,x_k}(Q_k^*)$ and an input database instance D. There exists an algorithm with preprocessing time $O(|D|)$ and space $O(|D|)$, such that $\pi_{x_1,\ldots,x_k}(Q_k^*)$ can be enumerated with delay $\delta = O\left(|\texttt{OUT}_\bowtie|/|\texttt{OUT}_\pi|^{1/k}\right)$ and space $S_e = O(|D|)$*

In the above theorem, we obtain delay guarantees that depend on both the full join result $\texttt{OUT}_\bowtie$ and the projection output size $\texttt{OUT}_\pi$.

However, one does not need to know $\texttt{OUT}_\bowtie$ or $\texttt{OUT}_\pi$ to apply the result. We first compare the result with Theorem 7. First, observe that both Theorem 10 and Theorem 7 require $O(|D|)$ preprocessing time. Second, the delay guarantee provided by Theorem 10 can be better than Theorem 7. This happens when $|\texttt{OUT}_\bowtie| \leq |D|\cdot J^{1-1/k}$, a condition that can be easily checked in linear time.

We now proceed to describe the algorithm. First, we compute all the statistics for computing J in linear time. If $J = |\mathbf{dom}(x_j)|$ for some integer $j \in \{1,2,\cdots,k\}$, we just materialize one result for each valuation of x_j. Otherwise, $J = \prod_{i=1}^{k}|\sigma_{y=u}R_i(x_i,y)|$ for some valuation u in attribute y. Note that we do need to explicitly materialize the cartesian product but only need to store the tuples in $\bigcup_{i\in\{1,2,\cdots,k\}}\sigma_{y=u}R_i(x_i,y)$. As mentioned before, each output in $\times_{i=1}^{k}(\pi_y\sigma_{y=u}R_i(x_i,y))$ can be enumerated with $O(1)$ delay. This preprocessing phase takes $O(|D|)$ time and $O(|D|)$ space. We can now invoke Lemma 5 to achieve the claimed delay. The final observation is to express J in terms of $|\texttt{OUT}_\pi|$. Note that $|\texttt{OUT}_\pi| \leq \Pi_{i\in[k]}|\mathbf{dom}(x_i)|$ which implies that $\max_{i\in[k]}|\mathbf{dom}(x_i)| \geq |\texttt{OUT}_\pi|^{1/k}$. Thus, it holds that $J \geq |\texttt{OUT}_\pi|^{1/k}$ which gives us the desired bound on the delay guarantee.

4.5 Fast Matrix Multiplication

Both Theorem 7 and Theorem 3 are combinatorial algorithms. In this section, we will show how fast matrix multiplication can be used to obtain a tradeoff between preprocessing time and delay that is better than Theorem 3 for some values of delay.

▶ **Theorem 11.** *Consider the star query $\pi_{x_1,\ldots,x_k}(Q_k^*)$ and an input database instance D. Then, there exists an algorithm that requires preprocessing $T_p = O((|D|/\delta)^{\omega+k-2})$ and can enumerate the query result with delay $O(\delta)$ for $1 \leq \delta \leq |D|^{(\omega+k-3)/(\omega+2\cdot k-3)}$.*

For the two-path query and the current best value of $\omega = 2.373$, we get the tradeoff as $T_p = O((|D|/\delta)^{2.373})$ and a delay guarantee of $O(\delta)$ for $|D|^{0.15} < \delta \leq |D|^{0.40}$. If we choose $\delta = |D|^{0.40}$, the preprocessing time is $T_p = O(|D|^{1.422})$. In contrast, Theorem 3 requires a preprocessing time of $T_p = O(|D|^{1.6})$, which is suboptimal compared to the above theorem. On the other hand, since $T_p = O(|D|^{1.422})$, we can safely assume that $|\texttt{OUT}_\bowtie| > |D|^{1.422}$, otherwise one can simply compute the full join in time $c^\star\cdot|D|^{1.422}$ using ListMerge, deduplicate and get constant delay enumeration. Applying Theorem 7 with $|\texttt{OUT}_\bowtie| > |D|^{1.422}$ tells us that we can obtain delay as $O(|D|^2/|\texttt{OUT}_\bowtie|) = O(|D|^{0.58})$. Thus, we can offer the user both choices and the user can decide which enumeration algorithm to use.

5 Left-Deep Hierarchical Queries

In this section, we will apply our techniques to another subset of hierarchical queries, which we call *left-deep*. A left-deep hierarchical query is of the following form:

$$Q^k_{\texttt{leftdeep}} = R_1(w_1, x_1) \bowtie R_2(w_2, x_1, x_2) \bowtie \ldots \bowtie R_{k-1}(w_{k-1}, x_1, \ldots, x_{k-1}) \bowtie R_k(w_k, x_1, \ldots, x_k)$$

It is easy to see that $Q^k_{\texttt{leftdeep}}$ is a hierarchical query for any $k \geq 1$. Note that for $k = 2$, we get the two-path query. For $k = 3$, we get $R(w_1, x_1) \bowtie S(w_2, x_1, x_2) \bowtie T(w_3, x_1, x_2)$. We will be interested in computing the query $\pi_{w_1,\ldots,w_k}(Q^k_{\texttt{leftdeep}})$, where we project out all the join variables. We show that the following result holds:

▶ **Theorem 12.** *Consider the query $\pi_{w_1,\ldots,w_k}(Q^k_{leftdeep})$ and any input database D. Then, there exists an algorithm that enumerates the query after preprocessing time $T_p = O(|D|)$ with delay $O(|D|^k/|\texttt{OUT}_\bowtie|)$.*

In the above theorem, $\texttt{OUT}_\bowtie$ is the full join result of the query $Q^k_{\texttt{leftdeep}}$ without projections. The AGM exponent for $Q^k_{\texttt{leftdeep}}$ is $\rho^* = k$. Observe that Theorem 12 is of interest when $|\texttt{OUT}_\bowtie| > |D|^{k-1}$ to ensure that the delay is smaller than $O(|D|)$. When the condition $|\texttt{OUT}_\bowtie| > |D|^{k-1}$ holds, the delay obtained by Theorem 12 is also better than the one given by the tradeoff in Theorem 3. In the worst-case when $|\texttt{OUT}_\bowtie| = \Theta(|D|^k)$, we can achieve constant delay enumeration after linear preprocessing time, compared to Theorem 3 that would require $\Theta(|D|^k)$ preprocessing time to achieve the same delay. The decision of when to apply Theorem 12 or Theorem 3 can be made in linear time by checking whether $|D|^k/|\texttt{OUT}_\bowtie|$ is smaller or larger than the actual delay guarantee obtained by the algorithm of Theorem 3 after linear time preprocessing.

6 Path Queries

In this section, we will study path queries. In particular, we will present an algorithm that enumerates the result of the query $\pi_{x_1,x_{k+1}}(P_k)$, i.e., the CQ that projects the two endpoints of a path query of length k. Recall that for $k \geq 3$, P_k is not a hierarchical query, and hence the tradeoff from [18] does not apply. A subset of path queries, namely 3-path and 4-path counting queries were considered in [17]. The algorithm used for counting the answers of 3-path and 4-path queries under updates constructed a set of views that can be used for the task of enumerating the query results under the static setting. Our result extends the same idea to apply to arbitrary length path queries, which we state next.

▶ **Theorem 13.** *Consider the query $\pi_{x_1,x_{k+1}}(P_k)$ with $k \geq 2$. For any input instance D and parameter $\epsilon \in [0, 1)$ there exists an algorithm that enumerates the query with preprocessing time (and space) $T_p = O(|D|^{2-\epsilon/(k-1)})$ and delay $O(|D|^\epsilon)$.*

We should note here that for $\epsilon = 1$, we can obtain a delay $O(|D|)$ using only linear preprocessing time $O(|D|)$ using the result of [3] since the query is acyclic, while for $\epsilon \to 1$ the above theorem would give preprocessing time $O(|D|^{2-1/(k-1)})$. Hence, for $k \geq 3$, we observe a discontinuity in the time-delay tradeoff. A second observation following from Theorem 13 is that as $k \to \infty$, the tradeoff collapses to only two extremal points: one where we get constant delay with $T_p = O(|D|^2)$, and the other where we get linear delay with $T_p = O(|D|)$.

7 Related Work

We overview prior work on static query evaluation for acyclic join-project queries. The result of any acyclic conjunctive query can be enumerated with constant delay after linear-time preprocessing if and only if it is free-connex [3]. This is based on the conjecture that Boolean multiplication of $n \times n$ matrices cannot be done in $O(n^2)$ time. Acyclicity itself is necessary for having constant delay enumeration: A conjunctive query admits constant delay enumeration after linear-time preprocessing if and only if it is free-connex acyclic [6]. This is based on a stronger hypothesis that the existence of a triangle in a hypergraph of n vertices cannot be tested in time $O(n^2)$ and that for any k, testing the presence of a k-dimensional tetrahedron cannot be tested in linear time. We refer the reader to an overview of pre-2015 for problems and progress related to constant delay enumeration [28]. Prior work also exhibits a dependency between the space and enumeration delay for conjunctive queries with access patterns [11]. It constructs a succinct representation of the query result that allows for enumeration of tuples over some variables under value bindings for all other variables. As noted by [18], it does not support enumeration for queries with free variables, which is also its main contribution. Our work demonstrates that for a subset of hierarchical queries, the tradeoff shown in [18] is not optimal. Our work introduces fundamentally new ideas that may be useful in improving the tradeoff for arbitrary hierarchical queries and enumeration of UCQs. There has also been some experimental work by the database community on problems related to enumerating join-project query results efficiently but without any formal delay guarantees. Seminal work [31, 30, 32, 12] has studied how compressed representations can be created apriori that allow for faster enumeration of query results. For the two path query, the fastest evaluation algorithm (with no delay guarantees) evaluates the projection join output in time $O(|D| \cdot |\mathtt{OUT}_\pi|^{\frac{(\omega-1)}{(\omega+1)}} + |D|^{\frac{2(\omega-1)}{(\omega+1)}} \cdot |\mathtt{OUT}_\pi|^{\frac{2}{(\omega+1)}})$ [9, 2]. For star queries, there is no closed form expression but fast matrix multiplication can be used to obtain instance dependent bounds on running time. Also related is the problem of dynamic evaluation of hierarchical queries. Recent work [17, 18, 4, 5] has studied the tradeoff between amortized update time and delay guarantees. Some of our techniques may also lead to new insights and improvements in existing algorithms. Prior work in differential privacy [26] and DGM [7] may also benefit from some of our techniques.

8 Conclusion and Open Problems

In this paper, we studied the problem of enumerating query results for an important subset of CQs with projections, namely star and path queries. We presented data-dependent algorithms that improve upon existing results by achieving non-trivial delay guarantees in linear preprocessing time. Our results are based on the idea of interleaving join query computation to achieve meaningful delay guarantees. Further, we showed how non-combinatorial algorithms (fast matrix multiplication) can be used for faster preprocessing to improve the tradeoff between preprocessing time and delay. We also presented new results on time-delay tradeoffs for a subset of non-hierarchical queries for the class of path queries. Our results also open several new tantalizing questions that open up possible directions for future work.

More preprocessing time for star queries. The second major open question is to show whether Theorem 7 can benefit from more preprocessing time to achieve lower delay guarantees. For instance, if we can afford the algorithm preprocessing time $T_p = O(|\mathtt{OUT}_\bowtie|/|D|^\epsilon + |D|)$ time, can we expect to get delay $\delta = O(|D|^\epsilon)$ for all $\epsilon \in (0, 1)$?

Sublinear delay guarantees for two-path query. It is not known whether we can achieve sublinear delay guarantee in linear preprocessing time for $Q_{\texttt{two-path}}$ query. This question is equivalent to the following problem: for what values of $|\texttt{OUT}_\pi|$ can $Q_{\texttt{path}}$ be evaluated in linear time. If $|\texttt{OUT}_\pi| = |D|^\epsilon$, then the best known algorithms can evaluate $Q_{\texttt{two-path}}$ in time $O(|D|^{1+\epsilon/3})$ (using fast matrix multiplication) [9] but this is still superlinear.

Space-delay bounds. The last question is to study the tradeoff between space vs delay for arbitrary hierarchical queries and path queries. Using some of our techniques, it may be possible to smartly materialize a certain subset of joins that could be used to achieve delay guarantees by interleaving with join computation. We also believe that the space-delay tradeoff implied by prior work can also be improved for certain ranges of delay by using the ideas introduced in this paper.

References

1 Mahmoud Abo Khamis, Hung Q Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 429–444, 2017.

2 Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *Proceedings of the 12th International Conference on Database Theory*, pages 121–126. ACM, 2009.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007.

4 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 303–318. ACM, 2017.

5 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering fo+ mod queries under updates on bounded degree databases. *ACM Transactions on Database Systems (TODS)*, 43(2):7, 2018.

6 Johann Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013.

7 Amrita Roy Chowdhury, Theodoros Rekatsinas, and Somesh Jha. Data-dependent differentially private parameter learning for directed graphical models. In *International Conference on Machine Learning*, pages 1939–1951. PMLR, 2020.

8 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Chapter 8.2, Introduction to algorithms*. MIT press, 2009.

9 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Fast join project query evaluation using matrix multiplication. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1213–1223, 2020.

10 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Enumeration algorithms for conjunctive queries with projection. *arXiv preprint arXiv:2101.03712*, 2021.

11 Shaleen Deep and Paraschos Koutris. Compressed representations of conjunctive query results. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 307–322. ACM, 2018.

12 Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. *To appear in Joint 2021 EDBT/ICDT Conferences, ICDT '21 Proceedings*, 2021.

13 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

14 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Treewidth and hypertree width. *Tractability: Practical Approaches to Hard Problems*, 1, 2014.

15 Gianluigi Greco and Francesco Scarcello. Structural tractability of enumerating csp solutions. *Constraints*, 18(1):38–74, 2013.

16 John E Hopcroft, Jeffrey D Ullman, and AV Aho. The design and analysis of computer algorithms, 1975.

17 Ahmet Kara, Hung Q Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Counting triangles under updates in worst-case optimal time. In *22nd International Conference on Database Theory*, 2019.

18 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 375–392, 2020.

19 Wojciech Kazana. *Query evaluation with constant delay*. PhD thesis, ENS Cachan, 2013.

20 Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 37–48. ACM, 2012.

21 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Record*, 42(4):5–16, 2013. doi:10.1145/2590989.2590991.

22 Dan Olteanu and Maximilian Schleich. Factorized databases. *ACM SIGMOD Record*, 45(2):5–16, 2016.

23 Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015.

24 Mark H Overmars and Jan Van Leeuwen. Dynamization of decomposable searching problems yielding good worst-case bounds. In *Theoretical Computer Science*, pages 224–233. Springer, 1981.

25 Anna Pagh and Rasmus Pagh. Scalable computation of acyclic joins. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 225–232, 2006.

26 Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. Crypt?: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 603–619, 2020.

27 Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Record*, 44(1):10–17, 2015. doi:10.1145/2783888.2783894.

28 Luc Segoufin. Constant delay enumeration for conjunctive queries. *ACM SIGMOD Record*, 44(1):10–17, 2015.

29 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases, synthesis lectures on data management. *Morgan & Claypool*, 2011.

30 Konstantinos Xirogiannopoulos and Amol Deshpande. Extracting and analyzing hidden graphs from relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 897–912. ACM, 2017.

31 Konstantinos Xirogiannopoulos, Udayan Khurana, and Amol Deshpande. Graphgen: Exploring interesting graphs in relational data. *Proceedings of the VLDB Endowment*, 8(12):2032–2035, 2015.

32 Konstantinos Xirogiannopoulos, Virinchi Srinivas, and Amol Deshpande. Graphgen: Adaptive graph processing using relational databases. In *Proceedings of the Fifth International Workshop on Graph Data-management Experiences & Systems*, GRADES'17, pages 9:1–9:7, New York, NY, USA, 2017. ACM. doi:10.1145/3078447.3078456.

33 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, volume 81, pages 82–94, 1981.

The Shapley Value of Inconsistency Measures for Functional Dependencies

Ester Livshits
Technion - Israel Institute of Technology, Haifa, Israel

Benny Kimelfeld
Technion - Israel Institute of Technology, Haifa, Israel

Abstract

Quantifying the inconsistency of a database is motivated by various goals including reliability estimation for new datasets and progress indication in data cleaning. Another goal is to attribute to individual tuples a level of responsibility to the overall inconsistency, and thereby prioritize tuples in the explanation or inspection of dirt. Therefore, inconsistency quantification and attribution have been a subject of much research in Knowledge Representation and, more recently, in Databases. As in many other fields, a conventional responsibility sharing mechanism is the Shapley value from cooperative game theory. In this paper, we carry out a systematic investigation of the complexity of the Shapley value in common inconsistency measures for functional-dependency (FD) violations. For several measures we establish a full classification of the FD sets into tractable and intractable classes with respect to Shapley-value computation. We also study the complexity of approximation in intractable cases.

2012 ACM Subject Classification Theory of computation → Incomplete, inconsistent, and uncertain databases; Information systems → Data cleaning

Keywords and phrases Shapley value, inconsistent databases, functional dependencies, database repairs

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.15

Related Version *Full Version*: `https://arxiv.org/abs/2009.13819`

Funding This work was supported by the Israel Science Foundation (ISF), Grant 768/19, and the German Research Foundation (DFG) Project 412400621 (DIP program).

1 Introduction

Inconsistency measures for knowledge bases have received considerable attention from the Knowledge Representation (KR) and Logic communities [9,13,15–17,19,20,37]. More recently, inconsistency measures have also been studied from the database viewpoint [2, 24]. Such measures quantify the extent to which the database violates a set of integrity constraints. There are multiple reasons why one might be using such measures. For one, the measure can be used for estimating the usefulness or reliability of new datasets for data-centric applications such as business intelligence [6]. Inconsistency measures have also been proposed as the basis of progress indicators for data-cleaning systems [24]. Finally, the measure can be used for attributing to individual tuples a level of responsibility to the overall inconsistency [30, 36], thereby prioritize tuples in the explanation/inspection/correction of errors.

A conventional approach to dividing the responsibility for a quantitative property (here an inconsistency measure) among entities (here the database tuples) is the *Shapley value* [35], which is a game-theoretic formula for wealth distribution in a cooperative game. The Shapley value has been applied in a plethora of domains, including economics [14], law [32], environmental science [22, 33], social network analysis [31], physical network analysis [29],

and advertisement [5]. In data management, the Shapley value has been used for determining the relative contribution of features in machine-learning predictions [21,28], the responsibility of tuples to database queries [4,23,34], and the reliability of data sources [6].

The Shapley value has also been studied in a context similar to the one we adopt in this paper – assigning a level of inconsistency to statements in an inconsistent knowledge base [17,30,36,39]. Hunter and Konieczny [15–17] use the maximal Shapley value of one inconsistency measure in order to define a new inconsistency measure. Grant and Hunter [12] considered information systems distributed along data sources of different reliabilities, and apply the Shapley value to determine the expected blame of each statement to the overall inconsistency. Yet, with all the investigation that has been conducted on the Shapley value of inconsistency, we are not aware of any results or efforts regarding the computational complexity of calculating this value.

In this work, we embark on a systematic analysis of the complexity of the Shapley value of database tuples relative to inconsistency measures, where the goal is to calculate the contribution of a tuple to inconsistency. Our main results are summarized in Table 1. We consider inconsistent databases with respect to functional dependencies (FDs), and basic measures of inconsistency following Bertossi [3] and Livshits, Ilyas, Kimelfeld and Roy [24]. We note that these measures are all adopted from the measures studied in the aforementioned KR research. In our setting, an individual tuple affects the inconsistency of only its containing relation, since the constraints are FDs. Hence, we focus on databases with a single relation. While most of our results extend to multiple relations, some extensions require a more subtle proof. We discuss the extension beyond a single relation in Section 8.

More formally, we investigate the following computational problem for any fixed combination of a relational signature, a set of FDs, and an inconsistency measure: given a database and a tuple, compute the Shapley value of the tuple with respect to the inconsistency measure. As Table 1 shows, two of these measures are computable in polynomial time: $\mathcal{I}_{\mathsf{MI}}$ (number of FD violations) and $\mathcal{I}_{\mathsf{P}}$ (number of problematic facts that participate in violations). For two other measures, we establish a full dichotomy in the complexity of the Shapley value: $\mathcal{I}_{\mathsf{d}}$ (the drastic measure – 0 for consistency and 1 for inconsistency) and $\mathcal{I}_{\mathsf{MC}}$ (number of maximal consistent subsets, a.k.a. repairs). The dichotomy in both cases is the same: when the FD set has, up to equivalence, an lhs chain (i.e., the left-hand sides form a chain w.r.t. inclusion [25]), the Shapley value can be computed in polynomial time; in any other case, it is $\mathrm{FP}^{\#\mathrm{P}}$-hard. In the case of $\mathcal{I}_{\mathsf{R}}$ (the minimal number of tuples to delete for consistency), the problem is solvable in polynomial time in the case of an lhs chain, and NP-hard whenever it is intractable to find a cardinality repair [27]; however, the problem is open for every FD set in between, for example, the bipartite matching constraint $\{A \to B, B \to A\}$.

We also study the complexity of approximating the Shapley value and show the following (as described in Table 1). First, in the case of $\mathcal{I}_{\mathsf{d}}$, there is a (multiplicative) fully polynomial-time approximation scheme (FPRAS) for every set of FDs. In the case of $\mathcal{I}_{\mathsf{MC}}$, approximating the Shapley value of *any* intractable (non-lhs-chain) FD set is at least as hard as approximating the number of maximal matchings of a bipartite graph – a long standing open problem [18]. In the case of $\mathcal{I}_{\mathsf{R}}$, we establish a full dichotomy, namely FPRAS vs. hardness of approximation, that has the same separation as the problem of finding a cardinality repair

The rest of this paper is organized as follows. After presenting the basic notation and terminology in Section 2, we formally define the studied problem and give initial observations in Section 3. In Section 4, we describe polynomial-time algorithms for $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{P}}$. Then, we explore the measures $\mathcal{I}_{\mathsf{d}}$, $\mathcal{I}_{\mathsf{R}}$ and $\mathcal{I}_{\mathsf{MC}}$ in Sections 5, 6 and 7, respectively. We conclude and discuss future directions in Section 8. Some proofs appear only in the archive version [26] and will be given in the full version of the paper.

Table 1 Complexity of the (exact ; approximate) Shapley value of inconsistency measures.

	lhs chain	no lhs chain, PTime c-repair	other
$\mathcal{I}_{\mathsf{d}}$	PTime	$\mathrm{FP}^{\#\mathrm{P}}$-complete ; FPRAS	
$\mathcal{I}_{\mathsf{MI}}$	PTime		
$\mathcal{I}_{\mathsf{P}}$	PTime		
$\mathcal{I}_{\mathsf{R}}$	PTime	**?** ; FPRAS	NP-hard [27] ; no FPRAS
$\mathcal{I}_{\mathsf{MC}}$	PTime	$\mathrm{FP}^{\#\mathrm{P}}$-complete [25] ; **?**	

2 Preliminaries

Database concepts. By a *relational schema* we refer to a sequence $(A_1, \ldots, A_n)$ of attributes. A database D over $(A_1, \ldots, A_n)$ is a finite set of tuples, or *facts*, of the form $(c_1, \ldots, c_n)$, where each c_i is a constant from a countably infinite domain. For a fact f and an attribute A_i, we denote by $f[A_i]$ the value associated by f with the attribute A_i (that is, $f[A_i] = c_i$). Similarly, for a sequence $X = (A_{j_1}, \ldots, A_{j_m})$ of attributes, we denote by $f[X]$ the tuple $(f[A_{j_1}], \ldots, f[A_{j_m}])$. Generally, we use letters from the beginning of the English alphabet (i.e., $A, B, C, \ldots$) to denote single attributes and letters from the end of the alphabet (i.e., $X, Y, Z, \ldots$) to denote sets of attributes. We may omit stating the relational schema of a database D when it is clear from the context or irrelevant.

A *Functional Dependency* (FD for short) over $(A_1, \ldots, A_n)$ is an expression of the form $X \to Y$, where $X, Y \subseteq \{A_1, \ldots, A_m\}$. We may also write the attribute sets X and Y by concatenating the attributes (e.g., $AB \to C$ instead of $\{A, B\} \to \{C\}$). A database D satisfies $X \to Y$ if every two facts $f, g \in D$ that agree on the values of the attributes of X also agree on the values of the attributes of Y (that is, if $f[X] = g[X]$ then $f[Y] = g[Y]$). A database D *satisfies* a set Δ of FDs, denoted by $D \models \Delta$, if D satisfies every FD of Δ. Otherwise, D *violates* Δ (denoted by $D \not\models \Delta$). Two FD sets over the same relational schema are *equivalent* if every database that satisfies one of them also satisfies the other.

Let Δ be a set of FDs and D a database (which may violate Δ). A *repair* (*of D w.r.t. Δ*) is a maximal consistent subset of D; that is, $E \subseteq D$ is a repair if $E \models \Delta$ but $E' \not\models \Delta$ for every $E \subsetneq E'$. A *cardinality repair* (or *c-repair* for short) is a repair of maximum cardinality; that is, it is a repair E such that $|E| \geq |E'|$ for every repair E'.

▶ **Example 1.** Figure 1 depicts an inconsistent database over a relational schema that stores a train schedule. For example, the fact f_1 states that train number 16 will depart from the New York Penn Station at time 1030 and arrive to the Boston Back Bay Station after 315 minutes. The FD set Δ consists of the two FDs:

- train time → departs
- train time duration → arrives

The first FD states that the departure station is determined by the train number and departure time, and the second FD states that the arrival station is determined by the train number, the departure time, and the duration of the ride.

Observe that the database of Figure 1 violates the FDs as all the facts refer to the same train number and departure time, but there is no agreement on the departure station. Moreover, the facts f_6 and f_7 also agree on the duration, but disagree on the arrival station. The database has five repairs: *(a)* $\{f_1, f_2\}$, *(b)* $\{f_3, f_4, f_5\}$, *(c)* $\{f_6, f_8\}$, *(d)* $\{f_7, f_8\}$, and *(e)* $\{f_9\}$; only the second one is a cardinality repair. ◊

fact	train	departs	arrives	time	duration
f_1	16	NYP	BBY	1030	315
f_2	16	NYP	PVD	1030	250
f_3	16	PHL	WIL	1030	20
f_4	16	PHL	BAL	1030	70
f_5	16	PHL	WAS	1030	120
f_6	16	BBY	PHL	1030	260
f_7	16	BBY	NYP	1030	260
f_8	16	BBY	WAS	1030	420
f_9	16	WAS	PVD	1030	390

Figure 1 The inconsistent database of our running example.

Shapley value. A *cooperative game* of a set A of players is a function $v : \mathcal{P}(A) \to \mathbb{R}$, where $\mathcal{P}(A)$ is the power set of A, such that $v(\emptyset) = 0$. The value $v(B)$ should be thought of as the joint wealth obtained by the players of B when they cooperate. The *Shapley value* of a player $a \in A$ measures the contribution of a to the total wealth $v(A)$ of the game [35], and is formally defined by

$$\text{Shapley}(A, v, a) \stackrel{\text{def}}{=} \frac{1}{|A|!} \sum_{\sigma \in \Pi_A} (v(\sigma_a \cup \{a\}) - v(\sigma_a))$$

where Π_A is the set of all permutations over the players of A and σ_a is the set of players that appear before a in the permutation σ. Intuitively, the Shapley value of a player a is the expected contribution of a to a subset constructed by drawing players randomly one by one (without replacement), where the contribution of a is the change to the value of v caused by the addition of a. An alternative formula for the Shapley value, that we will use in this paper, is the following.

$$\text{Shapley}(A, v, a) \stackrel{\text{def}}{=} \sum_{B \subseteq A \setminus \{a\}} \frac{|B|! \cdot (|A| - |B| - 1)!}{|A|!} \Big(v(B \cup \{a\}) - v(B) \Big)$$

Observe that $|B|! \cdot (|A| - |B| - 1)!$ is the number of permutations where the players of B appear first, then a, and then the rest of the players.

Approximation schemes. We discuss both exact and approximate algorithms for computing Shapley values. Recall that a *Fully-Polynomial Randomized Approximation Scheme* (FPRAS, for short) for a function f is a randomized algorithm $A(x, \epsilon, \delta)$ that returns an ϵ-approximation of $f(x)$ with probability at least $1 - \delta$, given an input x for f and $\epsilon, \delta \in (0, 1)$, in time polynomial in x, $1/\epsilon$, and $\log(1/\delta)$. Formally, an FPRAS, satisfies:

$$\Pr\left[f(x)/(1+\epsilon) \leq A(x, \epsilon, \delta) \leq (1+\epsilon)f(x)\right] \geq 1 - \delta\,.$$

Note that this notion of FPRAS refers to a *multiplicative* approximation, and we adopt this notion implicitly unless stated otherwise. We may also write "multiplicative" explicitly for stress. In cases where the function f has a bounded range, it also makes sense to discuss an *additive* FPRAS where $\Pr\left[f(x) - \epsilon \leq A(x, \epsilon, \delta) \leq f(x) + \epsilon\right] \geq 1 - \delta$. We refer to an additive FPRAS, and explicitly state so, in cases where the Shapley value is in the range $[0, 1]$.

3 The Shapley Value of Inconsistency Measures

In this paper, we study the Shapley value of facts with respect to measures of database inconsistency. More precisely, the cooperative game that we consider here is determined by an inconsistency measure $\mathcal{I}$, and the facts of the database take the role of the players. In turn, an *inconsistency measure* $\mathcal{I}$ is a function that maps pairs (D, Δ) of a database D and a set Δ of FDs to a number $\mathcal{I}(D, \Delta) \in [0, \infty)$. Intuitively, the higher the value $\mathcal{I}(D, \Delta)$ is, the more inconsistent (or, the less consistent) the database D is w.r.t. Δ. The Shapley value of a fact f of a database D w.r.t. an FD set Δ and inconsistency measure $\mathcal{I}$ is then defined as follows.

$$\text{Shapley}(D, \Delta, f, \mathcal{I}) \stackrel{\text{def}}{=} \sum_{E \subseteq (D \setminus \{f\})} \frac{|E|! \cdot (|D| - |E| - 1)!}{|D|!} \Big(\mathcal{I}(E \cup \{f\}, \Delta) - \mathcal{I}(E, \Delta) \Big) \quad (1)$$

We note that the definition of the Shapley value requires the cooperative game to be zero on the empty set [35] and this is indeed the case for all of the inconsistency measures $\mathcal{I}$ that we consider in this work. Next, we introduce each of these measures.

- $\mathcal{I}_{\mathsf{d}}$ is the *drastic measure* that takes the value 1 if the database is inconsistent and the value 0 otherwise [37].
- $\mathcal{I}_{\mathsf{MI}}$ counts the *minimal inconsistent subsets* [16, 17]; in the case of FDs, these subsets are simply the pairs of tuples that jointly violate an FD.
- $\mathcal{I}_{\mathsf{P}}$ is the number of *problematic facts*, where a fact is problematic if it belongs to a minimal inconsistent subset [10]; in the case of FDs, a fact is problematic if and only if it participates in a pair of facts that jointly violate Δ.
- $\mathcal{I}_{\mathsf{R}}$ is the minimal number of facts that we need to delete from the database for Δ to be satisfied (similarly to the concept of a cardinality repair and proximity in Property Testing) [3, 8, 11].
- $\mathcal{I}_{\mathsf{MC}}$ is the number of *maximal consistent subsets* (i.e., repairs) [10, 13].

Table 1 summarizes the complexity results for the different measures. The first column (lhs chain) refers to FD sets that have a left-hand-side chain – a notion that was introduced by Livshits et al. [25], and we recall in the next section. The second column (no lhs chain, PTime c-repair) refers to FD sets that do not have a left-hand-side chain, but entail a polynomial-time cardinality repair computation according to the dichotomy of Livshits et al. [27] that we discuss in more details in Section 6.

▶ **Example 2.** Consider again the database of our running example. Since the database is inconsistent w.r.t. the FD set defined in Example 1, we have that $\mathcal{I}_{\mathsf{d}}(D, \Delta) = 1$. As for the measure $\mathcal{I}_{\mathsf{MI}}$, the reader can easily verify that there are twenty eight pairs of tuples that jointly violate the FDs; hence, we have that $\mathcal{I}_{\mathsf{MI}}(D, \Delta) = 28$. Since each tuple participates in at least one violation of the FDs, it holds that $\mathcal{I}_{\mathsf{P}} = 9$. Finally, as we have already seen in Example 1, the database has five repairs and a single cardinality repair obtained by deleting six facts. Thus, $\mathcal{I}_{\mathsf{R}}(D, \Delta) = 6$ and $\mathcal{I}_{\mathsf{MC}}(D, \Delta) = 5$. In the next sections, we discuss the computation of the Shapley value for each one of these measures. ◊

Preliminary analysis. We study the *data complexity* of computing $\text{Shapley}(D, \Delta, f, \mathcal{I})$ for different inconsistency measures $\mathcal{I}$. To this end, we give here two important observations that we will use throughout the paper. The first observation is that the computation of $\text{Shapley}(D, \Delta, f, \mathcal{I})$ can be easily reduced to the computation of the expected value of

the inconsistency measure over all subsets of the database of a given size. In the following proposition, we denote by $\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}(D' \cup \{f\}, \Delta)\big)$ the expected value of $\mathcal{I}(D' \cup \{f\}, \Delta)$ over all subsets D' of $D \setminus \{f\}$ of a given size m, assuming a uniform distribution. Similarly, $\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}(D', \Delta)\big)$ is the expected value of $\mathcal{I}(D', \Delta)$ over all such subsets D'.

▶ **Proposition 3.** *Let $\mathcal{I}$ be an inconsistency measure. The following holds.*

$$\mathrm{Shapley}(D, \Delta, f, \mathcal{I}) = \frac{1}{|D|} \sum_{m=0}^{|D|-1} \Big[\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}(D' \cup \{f\}, \Delta)\big) - \mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}(D', \Delta)\big)\Big]$$

Proposition 3 implies that to compute the Shapley value of f, it suffices to compute the expectations of the amount of inconsistency over subsets D' and $D' \cup \{f\}$, where D' is drawn uniformly from the space of subsets of size m, for every m. More precisely, the computation of the Shapley value is Cook reducible[1] to the computation of these expectations. Our algorithms will, indeed, compute these expectations instead of the Shapley value.

The second observation is the following. One of the basic properties of the Shapley value is *efficiency* – the sum of the Shapley values over all the players equals the total wealth [35]. This property implies that $\sum_{f \in D} \mathrm{Shapley}(D, \Delta, f, \mathcal{I}) = \mathcal{I}(D, \Delta)$. Thus, whenever the measure itself is computationally hard, so is the Shapley value of facts.

▶ **Fact 1.** *Let $\mathcal{I}$ be an inconsistency measure. The computation of $\mathcal{I}$ is Cook reducible to the computation of the Shapley value of facts under $\mathcal{I}$.*

This observation can be used for showing lower bounds on the complexity of the Shapley value, as we will see in the next sections.

4 Tractable Measures: $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{P}}$

We start by discussing two tractable measures. The first measure is $\mathcal{I}_{\mathsf{MI}}$, that counts the *minimal inconsistent subsets* (i.e., pairs of facts that jointly violate at least one FD). An easy observation is that a fact f increases the value of the measure $\mathcal{I}_{\mathsf{MI}}$ by i in a permutation σ if and only if σ_f contains exactly i facts that are in conflict with f. Hence, assuming that D contains N_f facts that conflict with f, the Shapley value for this measure can be computed in the following way:

$$\begin{aligned}
\mathrm{Shapley}(D, \Delta, f, \mathcal{I}_{\mathsf{MI}}) &= \sum_{E \subseteq (D \setminus \{f\})} \frac{|E|! \cdot (|D| - |E| - 1)!}{|D|!} \Big(\mathcal{I}(E \cup \{f\}, \Delta) - \mathcal{I}(E, \Delta)\Big) \\
&= \frac{1}{|D|!} \sum_{i=1}^{N_f} \sum_{\substack{E \subseteq (D \setminus \{f\}) \\ |E \cap N_f| = i}} |E|! \cdot (|D| - |E| - 1)! \cdot i \\
&= \frac{1}{|D|!} \sum_{i=1}^{N_f} \sum_{m=i}^{|D|-1} \sum_{\substack{E \subseteq (D \setminus \{f\}) \\ |E| = m \\ |E \cap N_f| = i}} m! \cdot (|D| - m - 1)! \cdot i \\
&= \frac{1}{|D|!} \sum_{i=1}^{N_f} \sum_{m=i}^{|D|-1} \binom{N_f}{i} \binom{|D| - N_f - 1}{m - i} \cdot m! \cdot (|D| - m - 1)! \cdot i
\end{aligned}$$

[1] Recall that a *Cook reduction* from a function F to a function G is a polynomial-time *Turing reduction* from F to G, that is, an algorithm that computes F with an oracle to a solver of G.

Therefore, we immediately obtain the following result.

▶ **Theorem 4.** *Let Δ be a set of FDs. Computing* $\mathrm{Shapley}(D, \Delta, f, \mathcal{I}_{\mathsf{MI}})$ *can be done in polynomial time, given D and f.*

The second measure that we consider here is $\mathcal{I}_{\mathsf{P}}$ that counts the "problematic" facts; that is, facts that participate in a violation of Δ. Here, a fact f increases the measure by i in a permutation σ if and only if σ_f contains precisely $i-1$ facts that are in conflict with f, but not in conflict with any other fact of σ_f (hence, all these facts and f itself are added to the group of problematic facts). We prove the following.

▶ **Theorem 5.** *Let Δ be a set of FDs. Computing* $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{P}})$ *can be done in polynomial time, given D and f.*

Proof. We now show how the expected values of Proposition 3 can be computed in polynomial time. We start with $\mathbb{E}_{D' \sim U_m(D\setminus\{f\})}\big(\mathcal{I}_{\mathsf{P}}(D', \Delta)\big)$. We denote by X_m the random variable holding the number of problematic facts in a subset of size m of $D \setminus \{f\}$. We denote by X_m^g the random variable that holds 1 if the fact g is selected and participates in a violation of the FDs in such a subset, and 0 otherwise. In addition, we denote the expectations of these variables by $\mathbb{E}(X_m)$ and $\mathbb{E}(X_m^g)$, respectively (without explicitly stating the distribution $D' \sim U_m(D \setminus \{f\})$ in the subscript). Due to the linearity of expectation we have:

$$\mathbb{E}_{D' \sim U_m(D\setminus\{f\})}\big(\mathcal{I}_{\mathsf{P}}(D', \Delta)\big) = \mathbb{E}(X_m) = \mathbb{E}\Big(\sum_{g \in D\setminus\{f\}} X_m^g\Big) = \sum_{g \in D\setminus\{f\}} \mathbb{E}(X_m^g)$$

Hence, the computation of $\mathbb{E}_{D' \sim U_m(D\setminus\{f\})}\big(\mathcal{I}_{\mathsf{P}}(D', \Delta)\big)$ reduces to the computation of $\mathbb{E}(X_m^g)$, and this value can be computed as follows.

$$\begin{aligned}\mathbb{E}(X_m^g) = \Pr\left[g \text{ is selected in a subset of size } m\right] \times \\ \Pr\left[\text{a conflicting fact is selected in a subset of size } m \mid g \text{ is selected in the subset}\right] \\ = \frac{\binom{|D|-2}{m-1}}{\binom{|D|-1}{m}} \cdot \frac{\sum_{k=1}^{N_g}\binom{N_g}{k}\cdot\binom{|D|-1-N_g}{m-k-1}}{\binom{|D|-2}{m-1}} = \frac{\sum_{k=1}^{N_g}\binom{N_g}{k}\cdot\binom{|D|-1-N_g}{m-k-1}}{\binom{|D|-1}{m}}\end{aligned}$$

where N_g is the number of facts in $D \setminus \{f\}$ that are in conflict with g.

We can similarly show that $\mathbb{E}_{D' \sim U_m(D\setminus\{f\})}\big(\mathcal{I}_{\mathsf{P}}(D' \cup \{f\}, \Delta)\big) = \sum_{g \in D\setminus\{f\}} \mathbb{E}(X_{m,f}^g)$, where $X_{m,f}^g$ is a random variable that holds 1 if g is selected in a subset E of size m of $D \setminus \{f\}$ and participates in a violation of the FDs with the rest of the facts of $E \cup \{f\}$, and 0 otherwise. For a fact g that is not in conflict with f we have that: $\mathbb{E}(X_{m,f}^g) = \mathbb{E}(X_m^g)$, while for a fact g that is in conflict with f it holds that:

$$\mathbb{E}(X_{m,f}^g) = \Pr\left[g \text{ is selected in a subset of size } m\right] = \binom{|D|-2}{m-1} \Big/ \binom{|D|-1}{m}$$

and that concludes our proof. ◀

5 The Drastic Measure $\mathcal{I}_{\mathsf{d}}$

In this section, we consider the drastic measure $\mathcal{I}_{\mathsf{d}}$. While the measure itself is extremely simple and, in particular, computable in polynomial time (testing whether Δ is satisfied), it might be intractable to compute the Shapley value of a fact. In particular, we prove a dichotomy for this measure, classifying FD sets into ones where the Shapley value can be

computed in polynomial time and the rest where the problem is $\mathrm{FP}^{\#\mathrm{P}}$-complete.[2] Before giving our dichotomy, we recall the definition of a *left-hand-side chain* (lhs chain, for short), introduced by Livshits et al. [25].

▶ **Definition 6.** [25] *An FD set Δ has a left-hand-side chain if for every two FDs $X \to Y$ and $X' \to Y'$ in Δ, either $X \subseteq X'$ or $X' \subseteq X$.*

▶ **Example 7.** The FD set of our running example (Example 1) has an lhs chain. We could also define Δ with redundancy by adding the following FD: train time arrives $\to$ departs. The resulting FD set does not have an lhs chain, but it is *equivalent* to an FD set with an lhs chain. An example of an FD set that does not have an lhs chain, not even up to equivalence, is $\{\text{train time} \to \text{departs}, \text{train departs} \to \text{time}\}$. ◊

We prove the following.

▶ **Theorem 8.** *Let Δ be a set of FDs. If Δ is equivalent to an FD set with an lhs chain, then* $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{d}})$ *can be computed in polynomial time, given D and f. Otherwise, the problem is* $\mathrm{FP}^{\#\mathrm{P}}$*-complete.*

Interestingly, this is the exact same dichotomy that we obtained in prior work [25] for the problem of counting subset repairs. We also showed that this tractability criterion is decidable in polynomial time by computing a minimal cover: if Δ is equivalent to an FD set with an lhs chain, then every minimal cover of Δ has an lhs chain.

Proof of Theorem 8. The proof of the hardness side has two steps. We first show hardness for the matching constraint $\{A \to B, B \to A\}$ over the schema (A, B), and this proof is similar to the proof of Livshits et al. [23] for the problem of computing the Shapley contribution of facts to the result of the query $q() \coloneq R(x), S(x, y), T(y)$. Then, from this case to the remaining cases we apply the *fact-wise reductions* that we established in our prior work [25]. So, in the remainder of this section we will focus on the tractability side.

As stated in Proposition 3, the computation of $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{d}})$ reduces to the computation of the expected value of the measure over all subsets of the database of a given size m. In this case, $\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}_{\mathsf{d}}(D' \cup \{f\}, \Delta)\big)$ and $\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}_{\mathsf{d}}(D', \Delta)\big)$ are the probabilities that a uniformly chosen $D' \subseteq D \setminus \{f\}$ of size m is such that $(D' \cup \{f\}) \not\models \Delta$ and $D' \not\models \Delta$, respectively. Due to the structure of FD sets with an lhs chain, we can compute these probabilities efficiently, as we explain next.

Our main observation is that for an FD $X \to Y$, if we group the facts of D by X (i.e., split D into maximal subsets of facts that agree on the values of all attributes in X), then this FD and the FDs that appear later in the chain may be violated only among facts from the same group. Moreover, when we group by XY (i.e., further split each group of X into maximal subsets of facts that agree on the values of all attributes in Y), facts from different groups always violate this FD, and hence, violate Δ. We refer to the former groups as *blocks* and the latter groups as *subblocks*. This special structure allows us to split the problem into smaller problems, solve each one of them separately, and then combine the solutions via dynamic programming.

We define a data structure T where each vertex v is associated with a subset of D that we denote by $D[v]$. The root r is associated with D itself, that is, $D[r] = D$. At the first level, each child c of r is associated with a block of $D[r]$ w.r.t. $X_1 \to Y_1$, and each child c' of c is

[2] Recall that $\mathrm{FP}^{\#\mathrm{P}}$ is the class of polynomial-time functions with an oracle to a problem in #P (e.g., count the satisfying assignments of a propositional formula).

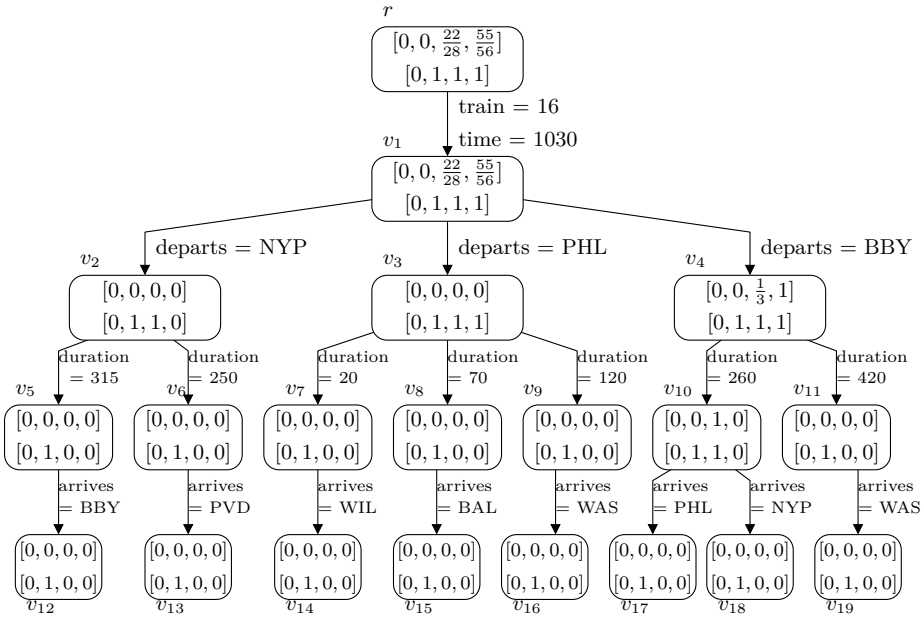


Figure 2 The data structure T of our running example.

associated with a subblock of $D[c]$ w.r.t. $X_1 \to Y_1$. At the second level, each child c'' of c' is associated with a block of $D[c']$ w.r.t. $X_2 \to Y_2$, and each child c''' of c'' is associated with a subblock of $D[c'']$ w.r.t. $X_2 \to Y_2$. This continues all the way to the nth FD, where at the ith level, each child u of an $(i-1)$th level subblock vertex v is associated with a block of $D[v]$ w.r.t. $X_i \to Y_i$ and each child u' of u is associated with a subblock of $D[u]$ w.r.t. $X_i \to Y_i$.

We assume that the data structure T is constructed in a preprocessing phase. Clearly, the number of vertices in T is polynomial in $|D|$ and n (recall that n is the number of FDs in Δ) as the height of the tree is $2n$, and each level contains at most $|D|$ vertices; hence, this preprocessing phase requires polynomial time (even under combined complexity). Then, we compute both $\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}_{\mathsf{d}}(D', \Delta)\big)$ and $\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}_{\mathsf{d}}(D' \cup \{f\}, \Delta)\big)$ by going over the vertices of T from bottom to top, as we will explain later. Note that for the computation of these values, we construct T from the database $D \setminus \{f\}$. Figure 2 depicts the data structure T used for the computation of $\mathrm{Shapley}(D, f_9, \Delta, \mathcal{I}_{\mathsf{d}})$ for the database D and fact f_9 of our running example. Next, we explain the meaning of the values stored in each vertex.

Each vertex v in T stores an array $v.\mathsf{val}$ with $|D[v]| + 1$ entries (that is initialized with zeros) such that $v.\mathsf{val}[j] = \mathbb{E}_{D' \sim U_j(D[v])}\big(\mathcal{I}_{\mathsf{d}}(D', \Delta)\big)$ for all $j \in \{0, \ldots, |D[v]|\}$ at the end of the execution. For this measure, we have that:

$$v.\mathsf{val}[j] \stackrel{\text{def}}{=} \Pr\left[\text{a random subset of size } j \text{ of } D[v] \text{ violates } \Delta\right]$$

Our final goal is to compute $r.\mathsf{val}[m]$, where r is the root of T. For that purpose, in the algorithm DrasticShapley, depicted in Figure 3, we go over the vertices of T in a bottom-up order and compute the values of $v.\mathsf{val}$ for every vertex v in the UpdateProb subroutine. Observe that we only need one execution of DrasticShapley with $m = |D| - 1$ to compute the required values for all $m \in \{1, \ldots, |D| - 1\}$, as we calculate all these values in our intermediate computations.

To compute $v.\mathsf{val}$ for a subblock vertex v, we iterate over its children in T (which are the $(i+1)$th level blocks) according to an arbitrary order defined in the construction of T. For a child c of v, we denote by $\mathsf{prev}(c)$ the set of children of v that occur before c in that order,

Algorithm 1 DrasticShapley(D, Δ, m, T).

1: **for all** vertices v of T in a bottom-up order **do**
2: UpdateProb(v, m)
3: **return** $r.\mathsf{val}[m]$

Subroutine 1 UpdateProb(v, m).

1: **for all** children c of v in T **do**
2: **for** $j \in \{m, \ldots, 1\}$ **do**
3: $$v.\mathsf{val}[j] = \sum_{\substack{j_1+j_2=j \\ 0\leq j_1\leq |D[c]| \\ 0\leq j_2\leq |D[\mathsf{prev}(c)]|}} \Big(c.\mathsf{val}[j_1] + (1 - c.\mathsf{val}[j_1]) \cdot v.\mathsf{val}[j_2]\Big) \cdot \frac{\binom{|D[c]|}{j_1}\cdot\binom{|D[\mathsf{prev}(c)]|}{j_2}}{\binom{|D[\mathsf{prev}(c)]|+|D[c]|}{j}}$$
4: **if** v is a block node **then**
5: $$v.\mathsf{val}[j] \mathrel{+}= \sum_{\substack{j_1+j_2=j \\ 0< j_1\leq |D[c]| \\ 0< j_2\leq |D[\mathsf{prev}(c)]|}} \Big((1 - c.\mathsf{val}[j_1]) \cdot (1 - v.\mathsf{val}[j_2])\Big) \cdot \frac{\binom{|D[c]|}{j_1}\cdot\binom{|D[\mathsf{prev}(c)]|}{j_2}}{\binom{|D[\mathsf{prev}(c)]|+|D[c]|}{j}}$$

Figure 3 An algorithm for computing $\mathbb{E}_{D'\sim U_m(D\setminus\{f\})}\big(\mathcal{I}_\mathsf{d}(D', \Delta)\big)$ for Δ with an lhs chain.

and by $D[\mathsf{prev}(c)]$ the database $\bigcup_{c'\in\mathsf{prev}(c)} D[c']$. When considering c in the for loop of line 1, we compute the expected value of the measure on a subset of $D[\mathsf{prev}(c)] \cup D[c]$. Hence, when we consider the last child of v in the for loop of line 1, we compute the expected value of the measure on a subset of the entire database $D[v]$.

For a child c of v, there are $N_1 = \binom{|D[\mathsf{prev}(c)]|+|D[c]|}{j}$ subsets of size j of all the children of v considered so far (including c itself). Each such subset consists of j_1 facts of the current c (there are $N_2 = \binom{|D[c]|}{j_1}$ possibilities) and j_2 facts of the previously considered children (there are $N_3 = \binom{|D[\mathsf{prev}(c)]|}{j_2}$ possibilities), for some j_1, j_2 such that $j_1 + j_2 = j$, with probability N_2N_3/N_1. Moreover, such a subset violates Δ if either the facts of the current c violate Δ (with probability $c.\mathsf{val}[j_1]$ that was computed in a previous iteration) or these facts satisfy Δ, but the facts of the previous children violate Δ (with probability $(1 - c.\mathsf{val}[j_1]) \cdot v.\mathsf{val}[j_2]$). Observe that since we go over the values j in reverse order in the for loop of line 2 (i.e., from m to 1), at each iteration of this loop, we have that $v.\mathsf{val}[j_2]$ (for all considered $j_2 \leq j$) still holds the expected value of $\mathcal{I}_\mathsf{d}$ over subsets of size j_2 of the previous children of v, which is indeed the value that we need for our computation.

This computation of $v.\mathsf{val}$ also applies to the block vertices. However, the addition of line 5 only applies to blocks. Since the children of a block belong to different subblocks, and two facts from the same ith level block but different ith level subbblocks always jointly violate $X_i \to Y_i$, a subset of size j of a block also violates the constraints if we select a non-empty subset of the current child c and a non-empty subset of the previous children, even if each of these subsets by itself is consistent w.r.t. Δ. Hence, we add this probability in line 5. Note that all the three cases that we consider are disjoint, so we sum the probabilities. Observe also that the leaves of T have no children and we do not update their probabilities, and, indeed the probability to select a subset from a leaf v that violates the constraints is zero, as all the facts of $D[v]$ agree on the values of all the attributes that occur in Δ.

To compute $\mathbb{E}_{D'\sim U_m(D\setminus\{f\})}\big(\mathcal{I}_\mathsf{d}(D'\cup\{f\}, \Delta)\big)$, we use the algorithm DrasticShapleyF, given in [26]. There, we distinguish between several types of vertices w.r.t. f, and show how this expectation can be computed for each one of these types.

▶ **Example 9.** We now illustrate the computation of $\mathbb{E}_{D' \sim U_m(D \setminus \{f_9\})}\big(\mathcal{I}_{\mathsf{d}}(D', \Delta)\big)$ on the database D and the fact f_9 of our running example for $m = 3$. Inside each node of the data structure T of Figure 2, we show the values $[v.\mathsf{val}[0], v.\mathsf{val}[1], v.\mathsf{val}[2], v.\mathsf{val}[3]]$ used for this computation. Below them, we present the corresponding values used in the computation of $\mathbb{E}_{D' \sim U_m(D \setminus \{f_9\})}\big(\mathcal{I}_{\mathsf{d}}(D' \cup \{f\}, \Delta)\big)$. For the leaves v and each vertex $v \in \{v_5, \ldots, v_9, v_{11}\}$, we have that $v.\mathsf{val}[j] = 0$ for every $j \in \{0, 1, 2, 3\}$, as $D[v]$ has a single fact. As for v_{10}, when we consider its first child v_{17} in the for loop of line 1 of UpdateProb, all the values in $v_{10}.\mathsf{val}$ remain zero (since $v_{17}.\mathsf{val}[j_1] = v_{10}.\mathsf{val}[j_2] = 0$ for any j_1, j_2, and $|D[\mathsf{prev}(c)]| = 0$). However, when we consider its second child v_{18}, while the computation of line 3 again has no impact on $v_{10}.\mathsf{val}$, after the computation of line 5 we have that $v_{10}.\mathsf{val}[2] = 1$. And, indeed, there is a single subset of size two of $D[v_{10}]$, which is $\{f_7, f_8\}$, and it violates the FD train time duration $\rightarrow$ arrives. This also affects the values of $v_4.\mathsf{val}$. In particular, when we consider the first child v_{10} of v_4, we have that $v_4.\mathsf{val}[j] = 1$ for $j = 2$ and $v_4.\mathsf{val}[j] = 0$ for any other j. Then, when we consider the second child v_{11} of v_4, it holds that $v_4.\mathsf{val}[2] = \frac{1}{3}$ (as the only subset of size two of $D[v_4]$ that violates the FDs is $\{f_6, f_7\}$, and there are three subsets in total) and $v_4.\mathsf{val}[3] = 1$ (as every subset of size three contains both f_6 and f_7). Finally, we have that $\mathbb{E}_{D' \sim U_3(D \setminus \{f_9\})}\big(\mathcal{I}_{\mathsf{d}}(D', \Delta)\big) = \frac{55}{56}$ and $\mathbb{E}_{D' \sim U_3(D \setminus \{f_9\})}\big(\mathcal{I}_{\mathsf{d}}(D' \cup \{f_9\}, \Delta)\big) = 1$. ◊

Approximation. We now consider an approximate computation of the Shapley value. Using the Chernoff-Hoeffding bound, we can easily obtain an additive FPRAS of the value $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{d}})$, by sampling $O(\log(1/\delta)/\epsilon^2)$ permutations and computing the average contribution of f in a permutation. As observed by Livshits et al. [23], a multiplicative FPRAS can be obtained using the same algorithm (possibly with a different number of samples) if the "gap" property holds: nonzero Shapley values are guaranteed to be large enough compared to the utility value (which is at most 1 in the case of the drastic measure). This is indeed the case here, as we now prove the following gap property of $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{d}})$.

▶ **Proposition 10.** *There is a polynomial p such that for all databases D and facts f of D the value* $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{d}})$ *is either zero or at least* $1/(p(|D|))$.

Proof. If no fact of D is in conflict with f, then $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{d}}) = 0$. Otherwise, let g be a fact that violates an FD of Δ jointly with f. Clearly, it holds that $\{g\} \models \Delta$, while $\{g, f\} \not\models \Delta$. The probability to choose a permutation σ, such that σ_f is exactly $\{g\}$ is $\frac{(|D|-2)!}{|D|!} = \frac{1}{|D| \cdot (|D|-1)}$ (recall that σ_f is the set of facts that appear before f in σ). Therefore, we have that $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{d}}) \geq \frac{1}{|D| \cdot (|D|-1)}$, and that concludes our proof. ◀

We conclude that the following holds.

▶ **Corollary 11.** $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{d}})$ *has both an additive and a multiplicative FPRAS.*

6 The Cost of a Cardinality Repair $\mathcal{I}_{\mathsf{R}}$

In this section, we study the measure $\mathcal{I}_{\mathsf{R}}$, based on the cost of a *cardinality repair* of D. Here, we refer to the number of facts that are removed from D to obtain a repair E as the cost of E. This is the first measure that we consider that is not always computable in polynomial time. Livshits et al. [27] established a dichotomy for the problem of computing a cardinality repair, classifying FD sets into those for which the problem is solvable in polynomial time, and those for which it is NP-hard. They presented a polynomial-time algorithm, which we refer to as Simplify, that takes as input an FD set Δ, finds a *removable* pair (X, Y) of attribute sets (if such a pair exists), and removes every attribute of $X \cup Y$ from every FD in

Algorithm 2 Simplify(Δ).

```
1: Remove trivial FDs from Δ
2: if Δ is not empty then
3:     find a removable pair (X, Y) of attribute sets
4:     Δ := Δ − XY
   return Δ
```

Figure 4 A simplification algorithm used for deciding whether a cardinality repair w.r.t. Δ can be computed in polynomial time [27].

Δ (we denote the result by $\Delta - XY$). A pair (X, Y) of attribute sets is considered removable if it satisfies the following three conditions:

- $\text{Closure}_\Delta(X) = \text{Closure}_\Delta(Y)$,
- XY is nonempty,
- every FD in Δ contains either X or Y on the left-hand side.

Note that it may be the case that $X = Y$, and then the conditions imply that every FD of Δ contains X on the left-hand side. The algorithm is depicted in Figure 4.

Livshits et al. [27] have shown that if it is possible to transform Δ to an empty set by repeatedly applying $\mathsf{Simplify}(\Delta)$, then a cardinality repair can be computed in polynomial time. Otherwise, the problem is NP-hard (and, in fact, APX-complete).

Fact 1 implies that computing $\text{Shapley}(D, f, \Delta, \mathcal{I}_\mathsf{R})$ is hard whenever computing $\mathcal{I}_\mathsf{R}(D, \Delta)$ is hard. Hence, we immediately obtain the following.

▶ **Theorem 12.** *Let Δ be a set of FDs. If Δ cannot be emptied by repeatedly applying* $\mathsf{Simplify}(\Delta)$*, then computing* $\text{Shapley}(D, f, \Delta, \mathcal{I}_\mathsf{R})$ *is NP-hard.*

In the remainder of this section, we focus on the tractable cases of the dichotomy of Livshits et al. [27]. In particular, we start by proving that the Shapley value can again be computed in polynomial time for an FD set that has an lhs chain.

▶ **Theorem 13.** *Let Δ be a set of FDs. If Δ is equivalent to an FD set with an lhs chain, then computing* $\text{Shapley}(D, f, \Delta, \mathcal{I}_\mathsf{R})$ *can be done in polynomial time, given D and f.*

Our polynomial-time algorithm RShapley, depicted in Figure 5, is very similar in structure to DrasticShapley. However, to compute the expected value of $\mathcal{I}_\mathsf{R}$, we take the reduction of Proposition 3 a step further, and show, that the problem of computing the expected value of the measure over subsets of size m can be reduced to the problem of computing the number of subsets of size m of D that have a cardinality repair of cost k, given m and k. In the subroutine UpdateCount, we compute this number.

For each vertex v in T, we define:

$$v.\mathsf{val}[j, t] \stackrel{\text{def}}{=} \text{number of subsets of size } j \text{ of } D[v] \text{ with a cardinality repair of cost } t$$

For the leaves v of T, we set $v.\mathsf{val}[j, 0] = \binom{|D[v]|}{j}$ for $0 \leq j \leq |D[v]|$, as every subset of $D[v]$ is consistent, and the cost of a cardinality repair is zero. We also set $v.\mathsf{val}[0, 0] = 1$ for each v in T for the same reason. Since the size of the cardinality repair is bounded by the size of the database, in $\mathsf{UpdateCount}(v, m)$, we compute the value $v.\mathsf{val}[j, t]$ for every $1 \leq j \leq m$ and $0 \leq t \leq j$. To compute this number, we again go over the children of v, one by one. When we consider a child c in the for loop of line 1, the value $v.\mathsf{val}[j, t]$ is the number of subsets of size j of $D[\mathsf{prev}(c)] \cup D[c]$ that have a cardinality repair of cost t.

Algorithm 3 $\mathsf{RShapley}(D, \Delta, m, T)$.

1: **for all** vertices v of T in a bottom-up order **do**
2: $\quad\mathsf{UpdateCount}(v, m)$
3: **return** $\sum_{k=0}^{m} \frac{k}{\binom{|D|-1}{m}} \cdot r.\mathsf{val}[m, k]$

Subroutine 2 $\mathsf{UpdateCount}(v, m)$.

1: $v.\mathsf{val}[0,0] = 1$
2: **if** v is a leaf **then** $v.\mathsf{val}[j,0] = \binom{|D[v]|}{j}$ for all $j \in \{1, \ldots, |D[v]|\}$
3: **for all** children c of v in T **do**
4: $\quad$**for** $j \in \{m, \ldots, 1\}$ **do**
5: $\quad\quad$**for** $t \in \{j, \ldots, 0\}$ **do**
6: $\quad\quad\quad$**if** v is a block vertex **then**
7: $\quad\quad\quad\quad v.\mathsf{val}[j,t] = \sum_{\substack{j_1+j_2=j \\ 0\le j_1\le|D[c]| \\ t-j_1\le j_2\le\min\{t,|D[\mathsf{prev}(c)]|\}}} \sum_{t-j_1\le w_2\le j_2} \left(c.\mathsf{val}[j_1, t-j_2]\cdot v.\mathsf{val}[j_2, w_2]\right)$
8: $\quad\quad\quad\quad v.\mathsf{val}[j,t] \mathrel{+}= \sum_{\substack{j_1+j_2=j \\ t-j_2\le j_1\le\min\{t,|D[c]|\} \\ 0\le j_2\le|D[\mathsf{prev}(c)]|}} \sum_{t-j_2< w_1\le j_1} \left(c.\mathsf{val}[j_1, w_1]\cdot v.\mathsf{val}[j_2, t-j_1]\right)$
9: $\quad\quad\quad$**else**
10: $\quad\quad\quad\quad v.\mathsf{val}[j,t] = \sum_{\substack{j_1+j_2=j \\ 0\le j_1\le|D[c]| \\ 0\le j_2\le|D[\mathsf{prev}(c)]|}} \sum_{\substack{t_1+t_2=t \\ 0\le t_1\le j_1 \\ 0\le t_2\le j_2}} \left(c.\mathsf{val}[j_1, t_1]\cdot v.\mathsf{val}[j_2, t_2]\right)$

Figure 5 An algorithm for computing $\mathbb{E}_{D'\sim U_m(D\setminus\{f\})}\left(\mathcal{I}_{\mathsf{R}}(D', \Delta)\right)$ for Δ with an lhs chain.

The children of a block v are subblocks that jointly violate an FD of Δ; hence, when we consider a child c of v, a cardinality repair of a subset E of $D[\mathsf{prev}(c)] \cup D[c]$ is either a cardinality repair of $E \cap D[c]$ (in which case we remove every fact of $E \cap D[\mathsf{prev}(c)]$) or a cardinality repair of $E \cap D[\mathsf{prev}(c)]$ (in which case we remove every fact of $E \cap D[c]$). The decision regarding which of these cases holds is based on the following four parameters: *(1)* the number j_1 of facts in $E \cap D[c]$, *(2)* the number j_2 of facts in $E \cap D[\mathsf{prev}(c)]$, *(3)* the cost w_1 of a cardinality repair of $E \cap D[c]$, and *(4)* the cost w_2 of a cardinality repair of $E \cap D[\mathsf{prev}(c)]$. In particular:

- If $w_1 + j_2 \le w_2 + j_1$, then a cardinality repair of $E \cap D[c]$ is preferred over a cardinality repair of $E \cap D[\mathsf{prev}(c)]$, as it requires removing less facts from the database.
- If $w_1 + j_2 > w_2 + j_1$, then a cardinality repair of $E \cap D[\mathsf{prev}(c)]$ is preferred over a cardinality repair of $E \cap D[c]$.

In fact, since we fix t in the computation of $v.\mathsf{val}[j,t]$, we do not need to go over all w_1 and w_2. In the first case, we have that $w_1 = t - j_2$ (hence, the total number of removed facts is $t - j_2 + j_2 = t$), and in the second case we have that $w_2 = t - j_1$ for the same reason. Hence, in line 7 we consider the first case where $t \le w_2 + j_1$, and in line 8 we consider the second case where $w_1 + j_2 > t$. To avoid negative costs, we add a lower bound of $t - j_1$ on j_2 and w_2 in line 7, and, similarly, a lower bound of $t - j_2$ on j_1 and w_1 in line 8.

For a subblock vertex v, a cardinality repair of $D[v]$ is the union of cardinality repairs of the children of v, as facts corresponding to different children of v do not jointly violate any FD. Therefore, for such vertices, in line 10, we compute $v.\mathsf{val}$ by going over all j_1, j_2 such that $j_1 + j_2 = j$ and all t_1, t_2 such that $t_1 + t_2 = t$ and multiply the number of subsets of size j_1 of the current child for which the cost of a cardinality repair is t_1 by the number of subsets of size j_2 of the previously considered children for which the cost of a cardinality repair is t_2.

We present a polynomial-time algorithm for computing $\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}_{\mathsf{R}}(D' \cup \{f\}, \Delta)\big)$ in [26].

Approximation. In cases where a cardinality repair can be computed in polynomial time, we can obtain an additive FPRAS in the same way as the drastic measure. (Note that this Shapley value is also in $[0, 1]$.) Moreover, we can again obtain a multiplicative FPRAS using the same technique due to the following gap property (proved similarly to Proposition 10).

▶ **Proposition 14.** *There is a polynomial p such that for all databases D and facts f of D the value* $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{R}})$ *is either zero or at least* $1/(p(|D|))$.

As aforementioned, Livshits et al. [27] showed that the hard cases of their dichotomy for the problem of computing a cardinality repair are, in fact, APX-complete; hence, there is a polynomial-time constant-ratio approximation, but for some $\epsilon > 1$ there is no (randomized) ϵ-approximation or else P = NP (NP $\subseteq$ BPP). Since the Shapley value of every fact w.r.t. $\mathcal{I}_{\mathsf{R}}$ is positive, the existence of a multiplicative FPRAS for $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{R}})$ would imply the existence of a multiplicative FPRAS for $\mathcal{I}_{\mathsf{R}}(D, \Delta)$ (due to Fact 1), which is a contradiction to the APX-hardness. We conclude the following.

▶ **Proposition 15.** *Let Δ be a set of FDs. If Δ can be emptied by repeatedly applying* $\mathsf{Simplify}(\Delta)$*, then* $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{R}})$ *has both an additive and a multiplicative FPRAS. Otherwise, it has neither multiplicative nor additive FPRAS, unless NP $\subseteq$ BPP.*

Unsolved cases for $\mathcal{I}_{\mathsf{R}}$. Unlike the other inconsistency measures considered in this paper, we do not have a full dichotomy for the measure $\mathcal{I}_{\mathsf{R}}$. In particular, a basic open problem is the computation of $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{R}})$ for $\Delta = \{A \to B, B \to A\}$. On the one hand, Proposition 15 shows that this case belongs to the tractable side if an approximation is allowed. On the other hand, our algorithm for exact $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{R}})$ is via counting the subsets of size m that have a cardinality repair of cost k. This approach will not work here:

▶ **Proposition 16.** *Let $\Delta = \{A \to B, B \to A\}$ be an FD set over (A, B). Counting the subsets of size m of a given database that have a cardinality repair of cost k is #P-hard.*

Proof. The proof is by a reduction from the problem of computing the number of perfect matchings in a bipartite graph, known to be #P-complete [38]. Given a bipartite graph $g = (A \cup B, E)$ (where $|A| = |B|$), we construct a database D over (A, B) by adding a fact (a, b) for every edge $(a, b) \in E$. We then define $m = |A|$ and $k = 0$. It is rather straightforward that the perfect matchings of g correspond exactly to the subsets D' of size $|A|$ of D such that D' itself is a cardinality repair. ◀

Observe that the cooperative game for $\Delta = \{A \to B, B \to A\}$ can be seen as a game on bipartite graphs where the vertices on the left-hand side represent the values of attribute A, the vertices on the right-hand side correspond to the values that occur in attribute B, and the edges represent the tuples of the database (hence, the players of the game). This game is different from the well-known matching game [1] where the players are the *vertices* of the graph (and the value of the game is determined by the maximum weight matching of the subgraph induced by the coalition). In contrast, in our case the players correspond to the *edges* of the graph. It is not clear what is the connection between the two games and whether or how we can use known results on matching games to derive results for the game that we consider here.

Algorithm 4 MCShapley(D, Δ, m, T).

for all vertices v of T in a bottom-up order **do**
    UpdateExpected(v, m)
return r.val$[m]$

Subroutine 3 UpdateExpected(v, m).

v.val$[0] = 1$
if v is a leaf **then** v.val$[j] = 1$ for all $j \in \{1, \ldots, |D[v]|\}$
for all children c of v in T **do**
    for $j \in \{m, \ldots, 1\}$ **do**
        if v is a block vertex **then**
            $$v.\mathsf{val}[j] = \sum_{\substack{j_1+j_2=j \\ 0\leq j_1\leq |D[c]| \\ 0\leq j_2\leq |D[\mathsf{prev}(c)]|}} \big(c.\mathsf{val}[j_1] + v.\mathsf{val}[j_2]\big)$$
        else
            $$v.\mathsf{val}[j] = \sum_{\substack{j_1+j_2=j \\ 0\leq j_1\leq |D[c]| \\ 0\leq j_2\leq |D[\mathsf{prev}(c)]|}} \big(c.\mathsf{val}[j_1] \cdot v.\mathsf{val}[j_2]\big)$$

Figure 6 An algorithm for computing $\mathbb{E}_{D'\sim U_m(D\setminus\{f\})}\big(\mathcal{I}_{\mathsf{MC}}(D', \Delta)\big)$ for Δ with an lhs chain.

7 The Number of Repairs $\mathcal{I}_{\mathsf{MC}}$

The final measure that we consider counts the repairs of the database. A dichotomy result from our previous work [25] states that the problem of counting repairs can be solved in polynomial time for FD sets with an lhs chain (up to equivalence), and is #P-complete for any other FD set. The hardness side, along with Fact 1, implies that computing Shapley$(D, f, \Delta, \mathcal{I}_{\mathsf{MC}})$ is $\mathrm{FP}^{\#\mathrm{P}}$-hard whenever the FD set is not equivalent to an FD set with an lhs chain. Hence, an lhs chain is a necessary condition for tractability. We show here that it is also sufficient: if the FD set has an lhs chain, then the problem can be solved in polynomial time. Consequently, we obtain the following dichotomy.

▶ **Theorem 17.** *Let Δ be a set of FDs. If Δ is equivalent to an FD set with an lhs chain, then computing* Shapley$(D, f, \Delta, \mathcal{I}_{\mathsf{MC}})$ *can be done in polynomial time, given D and f. Otherwise, the problem is* $\mathrm{FP}^{\#\mathrm{P}}$*-complete.*

The algorithm MCShapley, depicted in Figure 6, for computing Shapley$(D, f, \Delta, \mathcal{I}_{\mathsf{MC}})$, has the same structure as DrasticShapley, with the only difference being the computations in the subroutine UpdateExpected (that replaces UpdateProb).

For a vertex v in T we define:

$$v.\mathsf{val}[j] = \mathbb{E}\left[\text{number of repairs of a random subset of size } j \text{ of } D[v]\right]$$

As the number of repairs of a consistent database D is one (D itself is a repair), we set v.val$[0] = 1$ for every vertex v and v.val$[j] = 1$ for $0 \leq j \leq |D[v]|$ for every leaf v. Now, consider a block vertex v and a child c of v. Since the children of v are subblocks, each repair consists of facts of a single child. Hence, the total number of repairs is the sum of repairs of the children of v. Moreover, since our choice of facts from different subblocks is independent,

we have the following (where $\mathsf{MC}(D, \Delta)$ is the set of repairs of D w.r.t. Δ).

$$\mathbb{E}_{D' \sim U_j(D[\mathsf{prev}(c)] \cup D[c])}\big(\mathcal{I}_{\mathsf{MC}}(D', \Delta)\big) = \sum_{\substack{D' \subseteq D[\mathsf{prev}(c)] \cup D[c] \\ |D'| = j}} \Pr[D'] \cdot |MC(D', \Delta)| =$$

$$\sum_{\substack{j_1 + j_2 = j \\ 0 \leq j_1 \leq |D[c]| \\ 0 \leq j_2 \leq D[\mathsf{prev}(c)]}} \sum_{\substack{E_1 \subseteq D[c] \\ |E_1| = j_1}} \sum_{\substack{E_2 \subseteq D[\mathsf{prev}(c)] \\ |E_2| = j_2}} \Pr[E_1] \Pr[E_2] \big(|MC(E_1, \Delta)| + |MC(E_2, \Delta)|\big)$$

Using standard mathematical manipulations, we obtain the following result:

$$\mathbb{E}_{D' \sim U_j(D[\mathsf{prev}(c)] \cup D[c])}\big(\mathcal{I}_{\mathsf{MC}}(D', \Delta)\big) =$$
$$\sum_{\substack{j_1 + j_2 = j \\ 0 \leq j_1 \leq |D[c]| \\ 0 \leq j_2 \leq D[\mathsf{prev}(c)]}} \big[\mathbb{E}_{D' \sim U_{j_1}(D[c])}\big(\mathcal{I}_{\mathsf{MC}}(D', \Delta)\big) + \mathbb{E}_{D' \sim U_{j_2}(D[\mathsf{prev}(c)])}\big(\mathcal{I}_{\mathsf{MC}}(D', \Delta)\big)\big]$$

This calculation is reflected in line 6 of the UpdateExpected subroutine.

We can similarly show that for a subblock vertex v, it holds that:

$$\mathbb{E}_{D' \sim U_j(D[\mathsf{prev}(c)] \cup D[c])}\big(\mathcal{I}_{\mathsf{MC}}(D', \Delta)\big) =$$
$$\sum_{\substack{j_1 + j_2 = j \\ 0 \leq j_1 \leq |D[c]| \\ 0 \leq j_2 \leq D[\mathsf{prev}(c)]}} \big[\mathbb{E}_{D' \sim U_{j_1}(D[c])}\big(\mathcal{I}_{\mathsf{MC}}(D', \Delta)\big) \times \mathbb{E}_{D' \sim U_{j_2}(D[\mathsf{prev}(c)])}\big(\mathcal{I}_{\mathsf{MC}}(D', \Delta)\big)\big]$$

We use this result for the calculation of line 8. The difference between the calculations lies in the observation that the children of a subblock are blocks that never jointly violate Δ; hence, the number of repairs is obtained by multiplying the number of repairs of each child of v.

An algorithm for computing $\mathbb{E}_{D' \sim U_m(D \setminus \{f\})}\big(\mathcal{I}_{\mathsf{MC}}(D' \cup \{f\}, \Delta)\big)$ is given in [26].

Approximation. Repair counting for $\Delta = \{A \to B, B \to A\}$ is the problem of counting the maximal matchings of a bipartite graph. As the values $\mathrm{Shapley}(D, f, \Delta, \mathcal{I}_{\mathsf{MC}})$ are nonnegative and sum up to the number of repairs, we conclude that an FPRAS for Shapley implies an FPRAS for the number of maximal matchings. To the best of our knowledge, existence of the latter is a long-standing open problem [18]. This is also the case for any Δ' that is not equivalent to an FD set with an lhs chain, since there is a fact-wise reduction from Δ to such Δ' [25].

8 Conclusions

We studied the complexity of calculating the Shapley value of database facts for basic inconsistency measures, focusing on FD constraints. We showed that two of them are computable in polynomial time: the number of violations ($\mathcal{I}_{\mathsf{MI}}$) and the number of problematic facts ($\mathcal{I}_{\mathsf{P}}$). In contrast, each of the drastic measure ($\mathcal{I}_{\mathsf{d}}$) and the number of repairs ($\mathcal{I}_{\mathsf{MC}}$) features a dichotomy in complexity, where the tractability condition is the possession of an lhs chain (up to equivalence). For the cost of a cardinality repair ($\mathcal{I}_{\mathsf{R}}$) we showed a tractable fragment and an intractable fragment, but a gap remains on certain FD sets – the ones that do not have an lhs chain, and yet, a cardinality repair can be computed in polynomial time. We also studied the approximability of the Shapley value and showed, among other things, an FPRAS for $\mathcal{I}_{\mathsf{d}}$ and a dichotomy in the existence of an FPRAS for $\mathcal{I}_{\mathsf{R}}$.

This work has been restricted to schemas with a single relation. Some of the results immediately generalize to schemas with multiple relations. For instance, it is easy to show that all of our lower bounds generalize. It is also easy to show that our upper bounds generalize when considering the additive inconsistency measures $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_{\mathsf{P}}$ and $\mathcal{I}_{\mathsf{R}}$ (where the value of the measure on the entire database is the sum of the values over the individual relations of the schema). This is due to the linearity property of the Shapley value [35]: $\text{Shapley}(D, f, \Delta, a \cdot \alpha + b \cdot \beta) = a \cdot \text{Shapley}(D, f, \Delta, \alpha) + b \cdot \text{Shapley}(D, f, \Delta, \beta)$. We believe that our upper bounds can also be generalized to multiple relation schemas for the measures $\mathcal{I}_{\mathsf{d}}$ and $\mathcal{I}_{\mathsf{MC}}$, but this requires a more subtle proof that we leave for future work.

Many other directions are left open for future research. First, the picture is incomplete for the measure $\mathcal{I}_{\mathsf{R}}$. In particular, the complexity of the exact computation is open for the bipartite matching constraint $\{A \to B, B \to A\}$ that, unlike the known FD sets in the intractable fragment, has an FPRAS. In general, we would like to complete the picture of $\mathcal{I}_{\mathsf{R}}$ towards a full dichotomy. Moreover, for the schemas where there is no FPRAS for $\mathcal{I}_{\mathsf{R}}$, our results neither imply nor refute the existence of a constant-ratio approximation (for *some* constant). Second, the problems are immediately extendible to any type of constraints other than functional dependencies, such as denial constraints, tuple generating dependencies, and so on. Third, it would be interesting to see how the results extend to wealth distribution functions other than Shapley, e.g., the Banzhaff Power Index [7]. The tractable cases remain tractable for the Banzhaff Power Index, but it is not clear how (and whether) our proofs for the lower bounds generalize to this function. Finally, there is the practical question of implementation: while our algorithms terminate in polynomial time, we believe that they are hardly scalable without further optimization and heuristics ad-hoc to the use case; developing those is an important challenge for future research.

References

1 Haris Aziz and Bart de Keijzer. Shapley meets Shapley. In *STACS*, volume 25 of *LIPIcs*, pages 99–111. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

2 Leopoldo E. Bertossi. Measuring and computing database inconsistency via repairs. In *SUM*, volume 11142 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2018.

3 Leopoldo E. Bertossi. Repair-based degrees of database inconsistency. In *LPNMR*, volume 11481 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2019.

4 Leopoldo E. Bertossi and Floris Geerts. Data quality and explainable AI. *J. Data and Information Quality*, 12(2):11:1–11:9, 2020.

5 Omar Besbes, Antoine Désir, Vineet Goyal, Garud Iyengar, and Raghav Singal. Shapley meets uniform: An axiomatic framework for attribution in online advertising. In *WWW*, pages 1713–1723. ACM, 2019.

6 Laurence Cholvy, Laurent Perrussel, William Raynaut, and Jean - Marc Th 'e venin. Towards consistency-based reliability assessment. In *AAMAS*, pages 1643–1644. ACM, 2015.

7 Pradeep Dubey and Lloyd S. Shapley. Mathematical properties of the banzhaf power index. *Mathematics of Operations Research*, 4(2):99–131, 1979.

8 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

9 John Grant and Anthony Hunter. Measuring inconsistency in knowledgebases. *J. Intell. Inf. Syst.*, 27(2):159–184, 2006.

10 John Grant and Anthony Hunter. Measuring consistency gain and information loss in stepwise inconsistency resolution. In *ECSQARU*, volume 6717, pages 362–373. Springer, 2011.

11 John Grant and Anthony Hunter. Distance-based measures of inconsistency. In *ECSQARU*, volume 7958 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2013.

12 John Grant and Anthony Hunter. Using Shapley inconsistency values for distributed information systems with uncertainty. In *ECSQARU*, volume 9161 of *Lecture Notes in Computer Science*, pages 235–245. Springer, 2015.

13 John Grant and Anthony Hunter. Analysing inconsistent information using distance-based measures. *Int. J. Approx. Reasoning*, 89:3–26, 2017. doi:10.1016/j.ijar.2016.04.004.

14 Faruk Gul. Bargaining foundations of Shapley value. *Econometrica: Journal of the Econometric Society*, pages 81–95, 1989.

15 Anthony Hunter and Sébastien Konieczny. Shapley inconsistency values. In *KR*, pages 249–259. AAAI Press, 2006.

16 Anthony Hunter and Sébastien Konieczny. Measuring inconsistency through minimal inconsistent sets. In *KR*, pages 358–366. AAAI Press, 2008.

17 Anthony Hunter and Sébastien Konieczny. On the measure of conflicts: Shapley inconsistency values. *Artif. Intell.*, 174(14):1007–1026, 2010.

18 Yifan Jing and Akbar Rafiey. Counting maximal near perfect matchings in quasirandom and dense graphs. *CoRR*, abs/1807.04803, 2018.

19 Kevin M. Knight. Two information measures for inconsistent sets. *Journal of Logic, Language and Information*, 12(2):227–248, 2003.

20 Sébastien Konieczny, Jérôme Lang, and Pierre Marquis. Quantifying information and contradiction in propositional logic through test actions. In *IJCAI*, pages 106–111. Morgan Kaufmann, 2003.

21 Christophe Labreuche and Simon Fossier. Explaining multi-criteria decision aiding models with an extended Shapley value. In *IJCAI*, pages 331–339. ijcai.org, 2018.

22 Zhenliang Liao, Xiaolong Zhu, and Jiaorong Shi. Case study on initial allocation of shanghai carbon emission trading based on Shapley value. *Journal of Cleaner Production*, 103:338–344, 2015.

23 Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. The Shapley value of tuples in query answering. In *ICDT*, volume 155 of *LIPIcs*, pages 20: 1–20: 19. Schloss Dagstuhl, 2020.

24 Ester Livshits, Ihab F. Ilyas, Benny Kimelfeld, and Sudeepa Roy. Principles of progress indicators for database repairing. *CoRR*, abs/1904.06492, 2019.

25 Ester Livshits and Benny Kimelfeld. Counting and enumerating (preferred) database repairs. In *PODS*, pages 289–301. ACM, 2017.

26 Ester Livshits and Benny Kimelfeld. The Shapley value of inconsistency measures for functional dependencies. *CoRR*, abs/2009.13819, 2020.

27 Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing optimal repairs for functional dependencies. *ACM Trans. Database Syst.*, 45(1):4: 1–4: 46, 2020.

28 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NIPS*, pages 4765–4774, 2017.

29 Richard TB Ma, Dah Ming Chiu, John Lui, Vishal Misra, and Dan Rubenstein. Internet economics: The use of Shapley value for isp settlement. *IEEE/ACM Transactions on Networking (TON)*, 18(3):775–787, 2010.

30 Kedian Mu, Weiru Liu, and Zhi Jin. Measuring the blame of each formula for inconsistent prioritized knowledge bases. *Journal of Logic and Computation*, 22(3):481–516, February 2011. arXiv:https://academic.oup.com/logcom/article-pdf/22/3/481/3177718/exr002.pdf.

31 Ramasuri Narayanam and Yadati Narahari. A Shapley value-based approach to discover influential nodes in social networks. *IEEE Transactions on Automation Science and Engineering*, 8(1):130–147, 2011.

32 Tatiana Nenova. The value of corporate voting rights and control: A cross-country analysis. *Journal of financial economics*, 68(3):325–351, 2003.

33 Leon Petrosjan and Georges Zaccour. Time-consistent Shapley value allocation of pollution cost reduction. *Journal of economic dynamics and control*, 27(3):381–398, 2003.

34 Alon Reshef, Benny Kimelfeld, and Ester Livshits. The impact of negation on the complexity of the Shapley value in conjunctive queries. In *PODS*, pages 285–297. ACM, 2020.

35 Lloyd S Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker, editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton, 1953.

36 Matthias Thimm. Measuring inconsistency in probabilistic knowledge bases. In *UAI*, pages 530–537. AUAI Press, 2009.

37 Matthias Thimm. On the compliance of rationality postulates for inconsistency measures: A more or less complete picture. *KI*, 31(1):31–39, 2017.

38 L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

39 Bruno Yun, Srdjan Vesic, Madalina Croitoru, and Pierre Bisquert. Inconsistency measures for repair semantics in OBDA. In *IJCAI*, pages 1977–1983. ijcai.org, 2018.

Database Repairing with Soft Functional Dependencies

Nofar Carmeli
Technion - Israel Institute of Technology, Haifa, Israel

Martin Grohe
RWTH Aachen University, Germany

Benny Kimelfeld
Technion - Israel Institute of Technology, Haifa, Israel

Ester Livshits
Technion - Israel Institute of Technology, Haifa, Israel

Muhammad Tibi
Technion - Israel Institute of Technology, Haifa, Israel

Abstract

A common interpretation of soft constraints penalizes the database for every violation of every constraint, where the penalty is the cost (weight) of the constraint. A computational challenge is that of finding an optimal subset: a collection of database tuples that minimizes the total penalty when each tuple has a cost of being excluded. When the constraints are strict (i.e., have an infinite cost), this subset is a "cardinality repair" of an inconsistent database; in soft interpretations, this subset corresponds to a "most probable world" of a probabilistic database, a "most likely intention" of a probabilistic unclean database, and so on. Within the class of functional dependencies, the complexity of finding a cardinality repair is thoroughly understood. Yet, very little is known about the complexity of finding an optimal subset for the more general soft semantics. This paper makes a significant progress in this direction. In addition to general insights about the hardness and approximability of the problem, we present algorithms for two special cases: a single functional dependency, and a bipartite matching. The latter is the problem of finding an optimal "almost matching" of a bipartite graph where a penalty is paid for every lost edge and every violation of monogamy.

2012 ACM Subject Classification Theory of computation → Incomplete, inconsistent, and uncertain databases; Information systems → Data cleaning

Keywords and phrases Database inconsistency, database repairs, integrity constraints, soft constraints, functional dependencies

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.16

Funding This work was supported by the German Research Foundation (DFG) Project 412400621 (DIP program) and the Israel Science Foundation (ISF), Grant 768/19. The work of Nofar Carmeli was supported by the Google PhD Fellowship.

Acknowledgements The authors are very grateful to Alessio Conte and Peter Lindner for insightful discussions about the work described in this paper.

1 Introduction

Various challenges in data management are based on soft variants of database constraints (also referred to as *weak* or *approximate* constraints). In constraint discovery and mining, for instance, the goal is to find constraints, such as Functional Dependencies (FDs) [3, 8, 11] and beyond [2, 12, 16], that generally hold in the database but not necessarily in a perfect manner. There, the reason for the violations might be rare events (e.g., agreement on the zip code but

not the state) or noise (e.g., mistyping). Soft constraints also arise when reasoning about uncertain data [6, 9, 18, 19] – the database is viewed as a probabilistic space over possible worlds, and the violation of a weak constraint in a possible world is viewed as evidence that affects the world's probability.

Our investigation concerns the latter application of soft constraints. To be more precise, the semantics is that of a *parametric factor graph*: the probability of a possible world is the product of *factors* where every violation of the constraint contributes one factor; in turn, this factor is a weight that is assigned upfront to the constraint. This approach is highly inspired by successful concepts such as the Markov Logic Network (MLN) [17].[1] The computational problems are the typical ones of probabilistic modeling: marginal inference (compute the probability of a query answer) and maximum likelihood (find the most probable world) – the problem that we focus on here.

More specifically, we investigate the complexity of finding a most probable world in the case where the constraints are FDs. By taking the logarithms of the factors, this problem can be formally defined as follows. We are given a database D and a set Δ of FDs, where every tuple and every FD has a weight (which is a nonnegative number). We wish to obtain a cleaner subset E of D by deleting tuples. The cost of the subset E includes a penalty for every deleted tuple, and a penalty for every violation of (i.e., pair of tuples that jointly violate) an FD; the penalties are the weights of the tuple and the FD, respectively. The goal is to find a subset E with a minimal cost. In what follows, we refer to such E as an *optimal subset* and to the optimization problem of finding an optimal subset as *soft repairing*. The optimal subset corresponds to the "most likely intention" in the Probabilistic Unclean Database (PUD) framework of De Sa, Ilyas, Kimelfeld, Ré and Rekatsinas [18] in a restricted case that is studied in their work, and to the "most probable world" in the probabilistic database model of Sen, Deshpande and Getoor [19]. In the special case where the FDs are hard constraints (i.e., their weight is infinite or just too large to pay), an optimal subset is simply what is known as a "cardinality repair" [15] or, equivalently [14], a "most probable database" [6].

The computational challenge of soft repairing is that there are exponentially many candidate subsets. We investigate the data complexity of the problem, where the database schema and the FD set are fixed, and the input consists of the database D and all involved weights. Moreover, we assume that D consists of a single relation; this is done without loss of generality, since the problem boils down to soft repairing each relation independently (since an FD does not involve more than one relation).

The complexity of the problem is very well understood in the case of hard constraints (cardinality repairs). Gribkoff, Van den Broeck and Suciu [6] established complexity results for the case of unary FDs (having a single attribute on the left-hand side), and Livshits, Kimelfeld and Roy [14] completed the picture to a full (effective) dichotomy over all possible sets of FDs. For example, the problem is solvable in polynomial time for the FD sets $\{A \to B\}$, $\{A \to B, B \to A\}$ and $\{A \to B, B \to A, B \to C\}$, but is NP-hard for $\{A \to B, B \to C\}$. In contrast, very little is known about the more general case where the FDs are soft (and violations are allowed), where the problem seems to be fundamentally harder, both to solve and to reason about. Clearly, for every Δ where it is intractable to find a cardinality repair, the soft version is also intractable. But the other direction is false (under conventional complexity assumptions). For example, soft repairing is hard for $\Delta = \{A \to B, B \to A, B \to C\}$, for the

[1] More precisely, an MLN can be viewed as a database with weak constraints, where the set of facts includes all possible tuples over the (finite) domains of the attributes.

following reason. We can set the weights of $A \to B$ and $B \to C$ to be very high, making each of them a hard constraint in effect, and the weight of $B \to A$ very low, making it ignorable in effect. Hence, an optimal subset is a cardinality repair for $\{A \to B, B \to C\}$ that, as said above, is hard to compute.

So, which sets of FDs have a tractable soft repairing? The only polynomial-time algorithm we are aware of is that of De Sa et al. [18] for the special case of a single key constraint, that is, $\Delta = \{X \to Y\}$ where XY contain all of the schema attributes; they have left the more general case (that we study here) open. In this work, we make substantial progress in answering this question by presenting algorithms for two types of FD sets: *(a)* a single FD *and (b)* a matching constraint.

The first type generalizes the tractability of De Sa et al. [18] from a key constraint to an arbitrary FD (as long as it is the only FD in Δ). Like theirs, our algorithm employs dynamic programming, but in a more involved fashion. This is because their algorithm is based on the fact that in a key constraint $X \to Y$, any two tuples that agree on X are necessarily conflicting. We also show that our algorithm can be generalized to additional sets of FDs. For example, it turns out that the FD set $\{\text{name} \to \text{address}\,,\ \text{name address} \to \text{email}\}$ is tractable as well. (Note that the address attribute on the left-hand side of the second FD is not redundant, as in the ordinary semantics, since the FDs are treated as soft constraints.) In Section 4 we phrase the more general condition that this FD set satisfies.

The second type, matching constraints, refers to FD sets $\Delta = \{X \to Y, X' \to Y'\}$ over a schema with the attributes $A_1, \ldots, A_k$ where $X \cup Y = X' \cup Y' = X \cup X' = \{A_1, \ldots, A_k\}$ and there are no attributes other than $A_1, \ldots, A_k$. The simplest example is $\{A \to B, B \to A\}$ over the binary schema (A, B) that represents a bipartite graph, and the problem is that of finding the best "almost matching" of a bipartite graph where a penalty is paid for every lost edge and every violation of monogamy. A more involved example is $\{\text{fn ln} \to \text{addr}\,,\ \text{fn addr} \to \text{ln}\}$ over the schema $(\text{fn}, \text{ln}, \text{addr})$. Our algorithm is based on a reduction to the *Minimum Cost Maximum Flow* (MCMF) problem [4].

Whether our algorithms cover all of tractable cases remains an open problem for future investigation. (In the Conclusions we discuss the simplest FD sets where the question is left unsolved.) We do show, however, that there is a polynomial-time approximation algorithm with an approximation factor 3, that is, a subset where the penalty is at most three times the optimum.

The rest of the paper is organized as follows. We give the formal setup and the problem definition in Section 2. We then discuss the complexity of the general problem and its relationship to past results in Section 3. We describe our algorithm for soft repairing in Sections 4 and 5 for a single FD and a matching constraint, respectively, and conclude in Section 6.

2 Formal Setup

We begin with preliminary definitions and terminology that we use throughout the paper.

2.1 Databases, FDs and Repairs

A *relation schema* $R(A_1, \ldots, A_k)$ consists of a relation symbol R and a set $\{A_1, \ldots, A_k\}$ of attributes. A *database* D over R is a set of facts f of the form $R(c_1, \ldots, c_k)$, where each c_i is a *constant*. We denote by $f[A_i]$ the value that the fact f associates with attribute A_i (i.e., $f[A_i] = c_i$). Similarly, if $X = B_1 \cdots B_k$ is a sequence of attributes from $\{A_1, \ldots, A_k\}$, then

$f[X]$ is the tuple $(f[B_1], \ldots, f[B_k])$. We assume that every fact $f \in D$ is associated with a nonnegative weight, hereafter denoted w_f. (The weight of a fact is sometimes derived from a validity/existence probability [6, 19].)

A *Functional Dependency* (FD) over the relation schema $R(A_1, \ldots, A_k)$ is an expression φ of the form $X \to Y$ where $X, Y \subseteq \{A_1, \ldots, A_k\}$. A *violation* of an FD in a database D is a pair $\{f, g\}$ of tuples from D that agrees on the left-hand side (i.e., $f[X] = g[X]$) but disagrees on the right-hand side (i.e., $f[Y] \neq g[Y]$). An FD $X \to Y$ is *trivial* if $Y \subseteq X$. We denote by $\mathsf{vio}(D, \varphi)$ the set of all the violations of the FD φ in D. We say that D *satisfies* φ, denoted $D \models \varphi$, if it has no violations (i.e., $\mathsf{vio}(D, \varphi)$ is empty). The database D satisfies a set Δ of FDs, denoted by $D \models \Delta$, if D satisfies every FD in Δ; otherwise, D violates Δ (denoted $D \not\models \Delta$). We assume that every FD $\varphi \in \Delta$ has a nonnegative weight, that we denote by w_φ.

When there is no risk of ambiguity, we may omit the specification of the relation schema $R(A_1, \ldots, A_k)$ and simply assume that the involved databases and constraints are all over the same schema.

Let D be a database and let Δ be a set of FDs. A *repair* (*of D w.r.t. Δ*) is an inclusion-maximal consistent subset E; that is, $E \subseteq D$ and $E \models \Delta$, and moreover, $E' \not\models \Delta$ for every E' such that $E \subsetneq E' \subseteq D$. Note that the number of repairs can be exponential in the number of facts of D. A *cardinality repair* is a repair E of a maximal cardinality (i.e., $|E| \geq |E'|$ for every repair E').

2.2 Soft Constraints

We define the concept of *soft constraints* (or *weak constraints* or *weighted rules*) in the standard way of "penalizing" the database for every missing fact, on the one hand, and every violation, on the other hand. This is the concept adopted in past work such as the *parfactors* of De Sa et al. [18], the *soft keys* of Jha et al. [9], and the *PrDB* model of Sen et al. [19]. The concept can be viewed as a special case of the *Markov Logic Network* (MLN) [17].

Formally, let D be a database and Δ a set of FDs. The *cost* of a subset E of a database D is then defined as follows.

$$\mathsf{cost}(E \mid D) \stackrel{\text{def}}{=} \left(\sum_{f \in (D \setminus E)} w_f \right) + \left(\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)| \right) \tag{1}$$

As for the computational model, we assume that every weight is a rational number r/q that is represented using the numerator and the denominator, namely (r, q), where each of the two is an integer represented in the standard binary manner.

2.3 Problem Definition: Soft Repairing

The problem we study in this paper, referred to as *soft repairing*, is the optimization problem of finding a database subset with a minimal cost. Since we consider the data complexity of the problem, we associate with each relation schema and set of FDs a separate computational problem.

▶ **Problem 1** (Soft Repairing). *Let $R(A_1, \ldots, A_k)$ be a relation schema and Δ a set of FDs.* Soft repairing *(*for $R(A_1, \ldots, A_k)$ and Δ*) is the following optimization problem: Given a database D, find an optimal subset of D, that is, a subset E of D with a minimal* $\mathsf{cost}(E \mid D)$.

FLIGHTS

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2
UA123	Delta	01/01/2021	LA	NY	N652NW	1
DL456	Southwest	02/01/2021	NC	MA	N713DX	2
DL456	Southwest	03/01/2021	NJ	FL	N245DX	1
DL456	Delta	03/01/2021	CA	IL	N819US	4

(a) D.

FLIGHTS

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2
DL456	Southwest	02/01/2021	NC	MA	N713DX	2
DL456	Southwest	03/01/2021	NJ	FL	N245DX	1

(b) E_1.

FLIGHTS

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
DL456	Delta	03/01/2021	CA	IL	N819US	4

(c) E_2.

FLIGHTS

Flight	Airline	Date	Origin	Destination	Airplane	
UA123	United Airlines	01/01/2021	LA	NY	N652NW	3
UA123	United Airlines	01/01/2021	NY	UT	N652NW	2
DL456	Delta	03/01/2021	CA	IL	N819US	4

(d) E_3.

Figure 1 For the relation FLIGHTS(Flight, Airline, Date, Origin, Destination, Airplane) and the FDs Flight → Airline (with $w_{\varphi_1} = 5$) and Flight Airline Date → Destination (with $w_{\varphi_2} = 1$), a database D, a cardinality repair E_1, a weighted cardinality repair E_2, and an optimal subset E_3.

Note that a cardinality repair is an optimal subset in the special case where the weight w_φ of every FD φ is ∞ (or just higher than the cost of deleting the entire database), and the weight w_f of every fact f is 1. Livshits et al. [14] studied the complexity of finding a *weighted cardinality repair*, which is the same as a cardinality repair but the weight w_f of every fact f can be arbitrary. Hence, both types of cardinality repairs are consistent (i.e., the constraints are strictly satisfied). In contrast, an optimal subset in the general case may violate one or more of the FDs. In the next section we recall the known complexity results for cardinality and weighted cardinality repairs.

▶ **Example 2.** Our running example is based on the database of Figure 1 over the relation schema FLIGHTS(Flight, Airline, Date, Origin, Destination, Airplane) that contains information about domestic flights in the United States. The weight of each tuple appears on the rightmost column. The FD set Δ consists of the following FDs:

- Flight → Airline: a flight is associated with a single airline.
- Flight Airline Date → Destination: a flight on a certain date has a single destination.

We assume that the weight of the first FD is 5, and the weight of the second FD is 1 (as the same flight number can be reused for different flights).

The database E_1 of Figure 1 is a cardinality repair of D as no repair of D can be obtained by removing less then three facts. However, E_1 is not a weighted cardinality repair, since its

Algorithm 1 Simplify(Δ).

Remove trivial FDs from Δ
if Δ is not empty **then**
  find a removable pair (X, Y) of attribute set
  $\Delta := \Delta - XY$
return Δ

cost is eight, while the cost of E_2 is six. The reader can easily verify that E_2 is a weighted cardinality repair of D. Finally, E_3 is not a repair of D in the traditional sense as it contains a violation of the second FD, but it is an optimal subset of D with $\mathsf{cost}(E_3 \mid D) = 5$. ⌟

3 Preliminary Complexity Analysis

We consider the data complexity of the problem of computing an optimal subset. We assume that the schema and the set of FDs are fixed, and the input consists of the database. Livshits et al. [14] studied the problems of finding a cardinality repair and a weighted cardinality repair, and established a dichotomy over the space of all the sets of functional dependencies. In particular, they introduced an algorithm that, given a set Δ of FDs, decides whether:

1. A weighted cardinality repair can be computed in polynomial time; *or*
2. Finding a (weighted) cardinality repair is APX-complete.[2]

No other possibility exists. The algorithm is a recursive procedure that attempts to simplify Δ at each iteration by finding a *removable* pair (X, Y) of attribute sets, and removing every attribute of X and Y from all the FDs in Δ (which we denote by $\Delta - XY$). We say that a pair (X, Y) of attribute sets is removable if it satisfies the following properties:

- $\mathrm{Closure}_\Delta(X) = \mathrm{Closure}_\Delta(Y)$,
- XY is nonempty,
- every FD in Δ contains either X or Y on the left-hand side.

Note that the sets X and Y may be the same, and then the condition states that every FD contains X on the left-hand side.

The simplification procedure for an FD set Δ is depicted here as Algorithm 1. If we are able to transform Δ to an empty set of FDs by repeatedly applying simplifications, then the algorithm returns true and finding an optimal consistent subset is solvable in polynomial time. Otherwise, the algorithm returns false and the problem is APX-complete. We state their result for later reference.

▶ **Theorem 3.** [14] *Let Δ be a set of FDs. If Δ can be emptied via* Simplify(Δ) *steps, then a weighted cardinality repair can be computed in polynomial time; otherwise, finding a cardinality repair is APX-complete.*

The hardness side of Theorem 3 immediately implies the hardness of the more general soft-repairing problem. Yet, the other direction (tractability generalizes) is not necessarily true. As discussed in the Introduction, if $\Delta = \{A \to B, B \to A, B \to C\}$, then Δ, as a set of hard constraints, is classified as tractable according to Algorithm 1; however, this is not the case for soft constraints. We can generalize this example by stating that if Δ contains

[2] Recall that APX is the class of NP optimization problems that admit constant-ratio approximations in polynomial time. Hardness in APX is via the so called "PTAS" reductions (cf. textbooks on approximation complexity, e.g., [5]).

a *subset* that is hard according to Theorem 3, then soft repairing is hard. (This does not hold when considering only hard constraints, as the example shows that there exists an easy Δ with a hard subset.) In the following sections, we are going to discuss tractable cases of FD sets. Before that, we will show that the problem becomes tractable if one settles for an approximation.

3.1 Approximation

The following theorem shows that soft repairing admits a constant-ratio *approximation*, for the constant three, in polynomial time. This means that there is a polynomial-time algorithm for finding a subset with a cost of at most three times the minimum.

▶ **Theorem 4.** *For all FD sets, soft repairing admits a 3-approximation in polynomial time.*

Proof. We reduce soft repairing to the problem of finding a minimum weighted set cover where every element belongs to 3 sets. A simple greedy algorithm finds a 3-approximation to this problem in linear time [7].

We set the elements to be $\{(\{f, g\}, \delta) \mid f, g \in D, \delta \in \Delta, f \text{ and } g \text{ contradict } \delta\}$. Each element $(\{f, g\}, \delta)$ belongs to three sets: f with weight w_f, g with weight w_g, and $(\{f, g\}, \delta)$ with weight w_δ. Each minimal solution to this set cover problem can be translated to a soft repair: the selected sets that correspond to tuples are removed in the repair. Indeed, a minimal set cover of such a construction has to resolve each conflict by either paying for the removal of at least one of the tuples or paying for the violation. ◀

In terms of formal complexity, Theorem 4 implies that the problem of soft repairing is in APX (for every set of FDs). From this, from Theorem 3 and from the discussion that follows Theorem 3, we conclude the following.

▶ **Corollary 5.** *Let Δ be a set of FDs. Soft repairing for Δ is in APX. Moreover, if any subset of Δ cannot be emptied via* Simplify(Δ) *steps, then soft repairing is APX-complete for Δ.*

4 Algorithm for a Single Functional Dependency

In this section, we consider the case of a single functional dependency, and present a polynomial-time algorithm for soft repairing. Hence, we establish the following result.

▶ **Theorem 6.** *In the case of a single FD, soft repairing can be solved in polynomial time.*

Next, we prove Theorem 6 by presenting an algorithm. Later, we also generalize the argument and result beyond a single FD (Theorem 7).

We assume that the single FD is $\varphi \stackrel{\text{def}}{=} X \to Y$ and that our input database is D. We split D into *blocks* and *subblocks*, as we explain next. The blocks of D are the maximal subsets of D that agree on the X values. Denote these blocks by $D_1, \ldots, D_m$. Note that there are no conflicts across blocks; hence, we can solve the problem separately for each block and then an optimal subset E is simply the union of optimal subsets E_i of the blocks D_i:

$$E = \bigcup_{i=1}^{m} E_i$$

The subblocks of a block D_i are the maximal subsets of D_i that agree on the Y values (in addition to the X values). We denote these subblocks by $D_{i,1}, \ldots, D_{i,q_i}$. Note that two facts from the same subblock are consistent, while two facts from different subblocks are conflicting.

From here we continue with dynamic programming. For a number $j \in \{0, \ldots, q_i\}$, where q_i is the number of subblocks of D_i, and a number $k \in \{0, \ldots, |D_{i,1} \cup \cdots \cup D_{i,j}|\}$ of facts, we define the following values that we are going to compute:

- $C[i, j, k]$ is the cost of an optimal subset of $D_{i,1} \cup \cdots \cup D_{i,j}$ (i.e., the union of the first j subblocks) with precisely k facts.
- $F[i, j, k]$ is a subset of $D_{i,1} \cup \cdots \cup D_{i,j}$ that realizes $C[i, j, k]$, that is,

$$|F[i, j, k]| = k \quad \wedge \quad \mathsf{cost}\left(F[i, j, k] \mid D_{i,1} \cup \cdots \cup D_{i,j}\right) = C[i, j, k]$$

(If multiple choices of $F[i, j, k]$ exist, we select an arbitrary one.) Once we compute the $F[i, q_i, k]$, we are done since it then suffices to return the best subset over all k:

$$E_i = F[i, q_i, k] \text{ for } k = \operatorname*{argmin}_k C[i, q_i, k]$$

It remains to compute $C[i, j, k]$ and $F[i, j, k]$. We will focus on the former, as the latter is obtained by straightforward bookkeeping. The key observation is that if we decide to delete t facts from $D_{i,j}$, then we always prefer to delete the t facts with the minimal weight. We use this observation as follows.

For a subblock $D_{i,j}$ and $t \in \{0, \ldots, |D_{i,j}|\}$, denote by $\mathsf{top}(t, D_{i,j})$ an arbitrary subset of $D_{i,j}$ with t facts of the highest weight. Hence, $\mathsf{top}(t, D_{i,j})$ is obtained by taking a prefix of size t when sorting the tuples of $D_{i,j}$ from the heaviest to the lightest. Then $C[i, j, k]$ is computed as follows.

$$C[i, j, k] = \begin{cases} 0 & j = 0 \text{ and } k = 0; \\ \infty & j = 0 \text{ and } k > 0; \\ \min\limits_t \Big(C[i, j-1, k-t] + t(k-t)w_\varphi + \sum\limits_{\substack{f \in D_{i,j} \setminus \\ \mathsf{top}(t, D_{i,j})}} w_f \Big) & \text{otherwise.} \end{cases}$$

The correctness of the above computation is due to the definition of the cost in Equation (1). In particular, in the third case, we go over all options for the number t of facts taken from the subblock $D_{i,j}$ and choose an option with the minimum cost. This cost consists of the following components:

- $C[i, j-1, k-t]$ is the cost of the best choice of $k-t$ facts from the remaining $j-1$ subblocks.
- $t(k-t)w_\varphi$ is the cost of the violations in which the jth subblock participates: any combination of a fact from $D_{i,j}$ and a fact from the other subblocks is a violation of φ.
- $\sum_{f \in D_{i,j} \setminus \mathsf{top}(t, D_{i,j})} w_f$ is the cost of removing every fact that is not in $\mathsf{top}(t, D_{i,j})$ from the jth subblock.

This completes the description of the algorithm. From this description, the correctness should be a straightforward conclusion.

4.1 Generalization

We now show how the idea from the previous section can be generalized to some FD sets beyond singletons. An attribute A is an *lhs attribute* of an FD $X \to Y$ if $A \in X$, and it is a *consensus attribute* of $X \to Y$ if $X = \emptyset$ and $A \in Y$ (hence, $X \to Y$ states that all tuples should have the same A value). The simplification step of Algorithm 2 removes an attribute A if for every FD in Δ, it is either an lhs or a consensus attribute. We prove the following.

▶ **Theorem 7.** *Let Δ be a set of FDs. If Δ can be emptied via $\mathsf{L/C-Simplify}(\Delta)$ steps, then soft repairing for Δ is solvable in polynomial time.*

Algorithm 2 L/C-Simplify(Δ).

```
1: remove trivial FDs from Δ
2: if Δ is not empty then
3:     find A such that in each FD, A is either an lhs or a consensus attribute
4:     Δ := Δ − A
5: return Δ
```

Note that whenever Δ can be emptied via $\mathsf{L/C-Simplify}(\Delta)$ steps, it can also be emptied via $\mathsf{Simplify}(\Delta)$ steps. Indeed, if $\mathsf{L/C-Simplify}(\Delta)$ eliminates the attribute A, then we can take: *(a)* $X = \{A\}$ and $Y = \emptyset$ in Algorithm 1 if A is a consensus attribute of some FD, *or (b)* $X = Y = \{A\}$ if A is an lhs attribute of every FD. This is expected due to Theorems 3 and 7, and the observation of Section 3 that soft-repairing is hard whenever computing a cardinality repair is hard.

▶ **Example 8.** Consider the database and the FD set of our running example (Example 2). This FD set, which we denote here by Δ_1, can be emptied via $\mathsf{L/C-Simplify}(\Delta)$ steps, by selecting attributes in the following order:

$$\begin{aligned}
&\{\mathsf{Flight} \to \mathsf{Airline}\,,\ \mathsf{Flight\ Airline\ Date} \to \mathsf{Destination}\}\\
\mathsf{Flight}:&\{\emptyset \to \mathsf{Airline}\,,\ \mathsf{Airline\ Date} \to \mathsf{Destination}\}\\
\mathsf{Airline}:&\{\mathsf{Date} \to \mathsf{Destination}\}\\
\mathsf{Date}:&\{\emptyset \to \mathsf{Destination}\}\\
\mathsf{Destination}:&\{\}
\end{aligned}$$

Hence, Theorem 7 implies that soft repairing can be solved in polynomial time for Δ_1.

Next, consider the FD set Δ_2 consisting of the following FDs: $\mathsf{Flight} \to \mathsf{Airline}$ and $\mathsf{Flight\ Date} \to \mathsf{Destination}$. This FD set is logically equivalent to Δ_1; hence, they both entail the exact same cardinality repairs. However, these sets are no longer equivalent when considering soft repairing. In particular, two facts that agree on the values of the Flight and Date attributes, but disagree on the values of the Airline and Destination attributes, violate only one FD in Δ_1 but two FDs in Δ_2, which affects the cost of keeping these two tuples in the database. In fact, the FD set Δ_2 cannot be emptied via $\mathsf{L/C-Simplify}(\Delta)$ steps, as after removing the Flight attribute, no other attribute is either an lhs or a consensus attribute of the remaining FDs. The complexity of soft repairing for Δ_2 remains an open problem. ⌟

Next, we prove Theorem 7 by presenting a polynomial-time algorithm for soft repairing in the case where Δ can be emptied via $\mathsf{L/C-Simplify}(\Delta)$ steps. Our algorithm generalizes the idea of the algorithm for a single FD, and we again use dynamic programming.

The main observation is as follows. Let A be an attribute chosen by $\mathsf{L/C-Simplify}(\Delta)$, and let $D_1, \ldots, D_m$ be the maximal subsets of D that agree on the value of A, which we refer to as blocks (w.r.t. A). Two facts from different blocks violate *all* of the FDs wherein A is a consensus attribute and *none* of the FDs wherein A is an lhs attribute. Therefore, to compute the cost of a soft repair, each pair of facts from different blocks is charged with the violation of all FDs wherein A is a consensus attribute. Then, we can remove A from all FDs and continue the computation separately for each block.

Now, let Δ be an FD set that can be emptied via $\mathsf{L/C-Simplify}(\Delta)$ steps, and let $A_1, \ldots, A_n$ be the attributes in the order of such an elimination process. For each $\ell \in \{1, \ldots, n+1\}$, we denote by Δ_ℓ the FD set in line 2 of the ℓth iteration of this execution

(after removing the trivial FDs). Thus, Δ_1 contains every non-trivial FD of Δ, and Δ_{n+1} is empty. We also denote by w_ℓ the total weight of the FDs in Δ_ℓ of which A_ℓ is a consensus attribute (if there are no such FDs, then $w_\ell = 0$).

In the algorithm for a single FD, the recursion steps were with respect to the block D_i (which determines the value of X), and so the value of i was a parameter. Here, we need to maintain the assignment τ to all previously handled attributes, and we use τ and ℓ as parameters. Given $1 \leq \ell \leq n+1$, if τ is an assignment to the attributes $A_1, \ldots, A_{\ell-1}$, then D^τ denotes the database $\sigma_\tau D$ (i.e., the database that contains all the tuples that agree with τ on the values of the attributes $A_1, \ldots, A_{\ell-1}$). We denote by $D^\tau_1, \ldots, D^\tau_{q^\tau_\ell}$ the blocks of D^τ w.r.t. A_ℓ. Moreover, we denote by $\tau \wedge (A_\ell = j)$ the assignment to the attributes $A_1, \ldots, A_\ell$ that agrees with block D^τ_j on the value assigned to A_ℓ and agrees with τ on all other values. We denote by $F[\ell, \tau, j, k]$ an optimal subset of $D^\tau_1 \cup \cdots \cup D^\tau_j$ of size k w.r.t. Δ_ℓ. We also denote by $C[\ell, \tau, j, k]$ the cost of $F[\ell, \tau, j, k]$. According to Equation (1), our goal is to compute $F[1, \emptyset, q^\emptyset_1, k]$ for $k = \operatorname{argmin}_k C[1, \emptyset, q^\emptyset_1, k]$.

We again focus on the computation of $C[\ell, \tau, j, k]$ that can be done as follows.

$$C[\ell, \tau, j, k] = \begin{cases} \sum\limits_{f \in D^\tau \setminus \mathsf{top}(k, D^\tau)} w_f & \ell = n+1, \\ 0 & j = 0, k = 0, \\ \infty & j = 0, k > 0, \\ \min\limits_t \Big(C[\ell, \tau, j-1, k-t] + t(k-t)w_\ell + & \text{otherwise.} \\ \qquad C[\ell+1, \tau \wedge (A_\ell = j), q^{\tau \wedge (A_\ell = j)}_{\ell+1}, t] \Big) & \end{cases}$$

The first line (where $\ell = n+1$) refers to the case where Δ is empty. Since there are no FDs that need to be taken into account, the optimal subset of D^τ of size k consists of the k facts of the highest weight. In the fourth case, we go over all options for the number t of facts taken from the block D^τ_j and choose an option with the minimum cost. This cost consists of the following components:

- $C[\ell, \tau, j-1, k-t]$ is the cost of the best choice of $k-t$ facts from the remaining $j-1$ blocks.
- $t(k-t)w_\ell$ is the cost of the violations in which the jth block participates: any combination of a fact from D^τ_j and a fact from the other blocks $D^\tau_1 \cup \cdots \cup D^\tau_{j-1}$ is a violation of the FDs in which A_ℓ is a consensus attribute.
- $C[\ell+1, \tau \wedge (A_\ell = j), q^{\tau \wedge (A_\ell = j)}_{\ell+1}, t]$ is the cost of the further repairing needed following the elimination of A_ℓ (i.e., repairing with respect to $\Delta_{\ell+1}$) applied to the current block (the t facts from D^τ_j) .

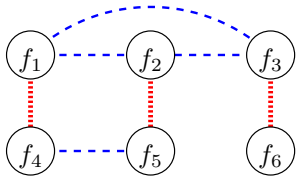
The given recursion can be computed in polynomial time via dynamic programming; thus, this proves Theorem 7.

5 Algorithm for Matching Constraints

In this section, we describe a polynomial-time algorithm for the special case of bipartite matching where the schema is $R(A, B)$ and $\Delta \stackrel{\text{def}}{=} \{A \to B, B \to A\}$. Note that each of the two FDs has a separate weight. In Section 5.1, we extend the algorithm into the more general case of (what we refer to as) a *matching constraint*, where the FD set Δ states two keys that cover all of the attributes. (We give the precise definition in Section 5.1.) We begin by proving the following lemma.

	A	B
f_1	a_1	b_1
f_2	a_1	b_2
f_3	a_1	b_3
f_4	a_2	b_1
f_5	a_2	b_2
f_6	a_3	b_3

(a) Database.



(b) Conflict graph.

Figure 2 A database over $R(A, B)$ and its conflict graph w.r.t. $\{A \to B, B \to A\}$.

▶ **Lemma 9.** *Soft repairing is solvable in polynomial time for* $R(A, B)$ *and* $\Delta = \{A \to B, B \to A\}$.

In the remainder of this section, we assume the input D over $R(A, B)$. We begin with an observation. For $E \subseteq D$ it holds that:

$$\sum_{f \in (D \setminus E)} w_f = \sum_{f \in D} w_f - \sum_{f \in E} w_f$$

Since the value $\sum_{f \in D} w_f$ does not depend on the choice of E, minimizing the value $\left(\sum_{f \in (D \setminus E)} w_f\right) + \left(\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)|\right)$ is the same as minimizing the value $\left(\sum_{f \in E} -w_f\right) + \left(\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)|\right)$. We use the following notation:

$$w_D(E) = \left(\sum_{f \in E} -w_f\right) + \left(\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)|\right)$$

To solve the problem, we construct a reduction to the *Minimum Cost Maximum Flow* (MCMF) problem. The input to MCMF is a flow network $\mathcal{N}$, that is, a directed graph (V, E) with a *source* node s having no incoming edges and a *sink* node t having no outgoing edges. Each edge $e \in E$ is associated with a *capacity* c_e and a *cost* $c(e)$. A flow f of $\mathcal{N}$ is a function $f : E \to \mathbb{R}$ such that $0 \leq f(e) \leq c_e$ for every $e \in E$, and moreover, for every node $v \in V \setminus \{s, t\}$ it holds that $\sum_{e \in I_v} f(e) = \sum_{e \in O_v} f(e)$ where I_v and O_v are the sets of incoming and outgoing edges of v, respectively. A *maximum flow* is a flow f that maximizes the value $\sum_{(s,v) \in E} f(s, v)$, and a *minimum cost maximum flow* is a maximum flow f with a minimal cost, where the cost of a flow is defined by $\sum_{e \in E} f(e) \cdot c(e)$. We say that f is *integral* if all values $f(e)$ are integers. It is known that, whenever the capacities are integral (i.e., natural numbers, as will be in our case), an integral minimum cost maximum flow exists and, moreover, can be found in polynomial time [1, Chapter 9].

From D we construct n instances $\mathcal{N}_1, \ldots, \mathcal{N}_n$ of the MCMF problem, where n is the number of facts in D, in the following way. First, we denote the FD $A \to B$ by φ_1 and the FD $B \to A$ by φ_2. We also denote by $D.A$ the set of values occurring in attribute A in D (that is, $D.A = \{\mathsf{a} \mid \exists f \in D(f[A] = \mathsf{a})\}$). We do the same for attribute B and denote by $D.B$ the set of values that occur in attribute B in D. For each value $\mathsf{a} \in D.A$ we denote by $\#_{D.A}(\mathsf{a})$ the number of appearances of the value a in attribute A (i.e., the number of facts $f \in D$ such that $f[A] = \mathsf{a}$). Similarly, we denote by $\#_{D.B}(\mathsf{b})$ the number of appearances of the value b in attribute B in D. Observe that

$$\mathsf{vio}(D, \varphi_1) = \frac{1}{2} \cdot \sum_{\mathsf{a} \in D.A} [\#_{D.A}(\mathsf{a}) \cdot (\#_{D.A}(\mathsf{a}) - 1)]$$

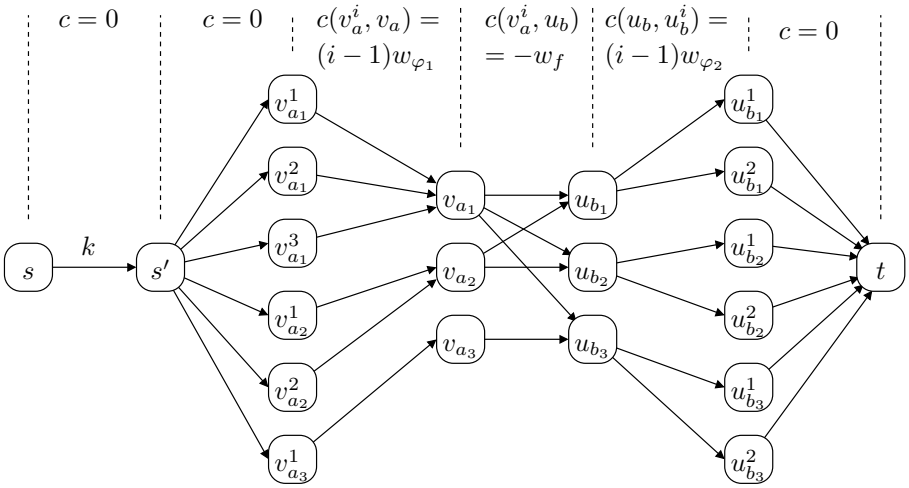


Figure 3 The network $\mathcal{N}_k$ constructed from the database of Figure 2a. The capacity of all edges is 1, except for the edge (s, s') that has capacity k.

since every fact of the form $R(\mathtt{a}, \mathtt{b})$ violates φ_1 with every fact $R(\mathtt{a}, \mathtt{c})$ where $\mathtt{b} \neq \mathtt{c}$. Similarly, it holds that

$$\mathsf{vio}(D, \varphi_2) = \frac{1}{2} \cdot \sum_{\mathtt{b} \in D.B} [\#_{D.B}(\mathtt{b}) \cdot (\#_{D.B}(\mathtt{b}) - 1)]$$

Next, we describe the construction of the network $\mathcal{N}_k$. Our construction for the database of Figure 2a is illustrated in Figure 3. Note that Figure 2b depicts the conflict graph of the database of Figure 2a w.r.t. $\Delta = \{A \to B, B \to A\}$, which contains a vertex for each fact in the database and an edge between two vertices if the corresponding facts jointly violate an FD of Δ. The blue edges in the conflict graph are violations of the FD $A \to B$ and the red edges are violations of the FD $B \to A$.

For each $k \in \{1, \ldots, n\}$ we construct the network $\mathcal{N}_k$ that consists of the set $\{s, s', t\} \cup V \cup A \cup B \cup U$ of nodes where:

- $A = \{v_{\mathtt{a}} \mid \mathtt{a} \in D.A\}$
- $B = \{u_{\mathtt{b}} \mid \mathtt{b} \in D.B\}$
- $V = \{v_{\mathtt{a}}^i \mid \mathtt{a} \in D.A, 1 \leq i \leq \#_{D.A}(\mathtt{a})\}$
- $U = \{u_{\mathtt{b}}^i \mid \mathtt{b} \in D.B, 1 \leq i \leq \#_{D.B}(\mathtt{b})\}$

$\mathcal{N}_k$ contains the following edges:

- (s, s'), with cost $c(s, s') = 0$
- $(s', v_{\mathtt{a}}^i)$ for every $v_{\mathtt{a}}^i \in V$, with cost $c(s', v_{\mathtt{a}}^i) = 0$
- $(v_{\mathtt{a}}^i, v_{\mathtt{a}})$ for every value $\mathtt{a} \in D$, with cost $c(v_{\mathtt{a}}^i, v_{\mathtt{a}}) = (i-1) \cdot w_{\varphi_1}$
- $(v_{\mathtt{a}}, u_{\mathtt{b}})$ for every $\mathtt{a} \in D.A$ and $\mathtt{b} \in D.B$ such that $f = R(\mathtt{a}, \mathtt{b})$ occurs in D, with cost $c(v_{\mathtt{a}}, u_{\mathtt{b}}) = -w_f$
- $(u_{\mathtt{b}}, u_{\mathtt{b}}^i)$ for every value $\mathtt{b} \in D$, with cost $c(u_{\mathtt{b}}, u_{\mathtt{b}}^i) = (i-1) \cdot w_{\varphi_2}$
- $(u_{\mathtt{b}}^i, t)$ for every $u_{\mathtt{b}}^i \in U$, with cost $c(u_{\mathtt{b}}^i, t) = 0$

The capacity of the edge (s, s') is k and the capacity of the other edges is 1. The intuition for the construction is as follows. A network with edges of the form $(v_{\mathtt{a}}, u_{\mathtt{b}})$ that are connected to a source on one side and a target on the other corresponds to a matching, which in turn corresponds to a traditional repair. To allow violations of $A \to B$, we add the vertices $v_{\mathtt{a}}^i$.

The cost of a violation of this FD is defined by the cost of the edges $(v^i_{\mathtt{a}}, v_{\mathtt{a}})$. In particular, if we keep k facts of the form $R(\mathtt{a}, \cdot)$ for some $\mathtt{a} \in D.A$ we pay $\sum_{i=1}^{k}(i-1)w_{\varphi_1}$ for violations of φ_1. We include the vertices $u^i_{\mathtt{b}}$ to similarly allow violations of $B \to A$. The discarding of facts is discouraged by offering gain for the edges $(v_{\mathtt{a}}, u_{\mathtt{b}})$. Finally, to prevent the case where the flow always fills the entire network (which corresponds to taking all facts and paying for all violations), we introduce the edge (s, s') which limits the capacity of the network, and enables us to find the minimum cost flow of a given size k. We will show that for every k, the cost of the solution to the MCMF problem on $\mathcal{N}_k$ will be the cost of the "cheapest" subinstance of D of size k. Hence, the solution to our problem is the cost of the minimal solution among all the instances $\mathcal{N}_1, \ldots, \mathcal{N}_n$.

Given an integral flow f in $\mathcal{N}_k$, the repair $D[f]$ induced by f, is the set of facts $R(\mathtt{a}, \mathtt{b})$ corresponding to edges of the form $(v_{\mathtt{a}}, u_{\mathtt{b}})$ such that $f(v_{\mathtt{a}}, u_{\mathtt{b}}) = 1$. Moreover, given a subinstance E of D of size k, we denote by f_E the integral flow in $\mathcal{N}_k$ defined as follows.

- $f_E(s, s') = k$
- $f_E(s', v^i_{\mathtt{a}}) = 1$ for $1 \leq i \leq \#E.A(\mathtt{a})$ and $f_E(s', v^i_{\mathtt{a}}) = 0$ for $i > \#E.A(\mathtt{a})$ for every $\mathtt{a} \in E.A$
- $f_E(v^i_{\mathtt{a}}, v_{\mathtt{a}}) = 1$ for $1 \leq i \leq \#E.A(\mathtt{a})$ and $f_E(v^i_{\mathtt{a}}, v_{\mathtt{a}}) = 0$ for $i > \#E.A(\mathtt{a})$ for every $\mathtt{a} \in E.A$
- $f_E(v_{\mathtt{a}}, u_{\mathtt{b}}) = 1$ if $R(\mathtt{a}, \mathtt{b}) \in E$ and $f_E(v_{\mathtt{a}}, u_{\mathtt{b}}) = 0$ otherwise
- $f_E(u_{\mathtt{b}}, u^i_{\mathtt{b}}) = 1$ for $1 \leq i \leq \#E.B(\mathtt{b})$ and $f_E(u_{\mathtt{b}}, u^i_{\mathtt{b}}) = 0$ for $i > \#E.B(\mathtt{b})$ for every $\mathtt{b} \in E.B$
- $f_E(u^i_{\mathtt{b}}, t) = 1$ for $1 \leq i \leq \#E.B(\mathtt{b})$ and $f_E(u^i_{\mathtt{b}}, t) = 0$ for $i > \#E.B(\mathtt{b})$ for every $\mathtt{b} \in E.B$

The reader can easily verify that f_E is indeed an integral flow in $\mathcal{N}_k$. Clearly, the value of the flow is k.

We have the following lemma.

▶ **Lemma 10.** *Every integral solution f to MCMF on $\mathcal{N}_k$ satisfies* $\mathsf{cost}(f) = w_D(f[D])$.

Proof. First, note that it cannot be the case that $f(s', v^j_{\mathtt{a}}) = 0$ while $f(s', v^i_{\mathtt{a}}) = 1$ for some $j < i$ and $i \in \{1, \ldots, \#_{D.A}(\mathtt{a})\}$. Otherwise, we can construct a different integral flow f' with $f'(s', v^j_{\mathtt{a}}) = f'(v^j_{\mathtt{a}}, v_{\mathtt{a}}) = 1$, $f'(s', v^i_{\mathtt{a}}) = f'(v^i_{\mathtt{a}}, v_{\mathtt{a}}) = 0$, and $f'(e) = f(e)$ for every other edge e. It holds that $\mathsf{cost}(f') = \mathsf{cost}(f) - c(v^i_{\mathtt{a}}, v_{\mathtt{a}}) + c(v^j_{\mathtt{a}}, v_{\mathtt{a}})$, and since $c(v^i_{\mathtt{a}}, v_{\mathtt{a}}) > c(v^j_{\mathtt{a}}, v_{\mathtt{a}})$ we will have that $\mathsf{cost}(f') < \mathsf{cost}(f)$ in contradiction to the fact that f is a solution to MCMF on $\mathcal{N}_k$. Therefore, for every $\mathtt{a} \in D.A$, if the flow entering the node $v_{\mathtt{a}}$ is ℓ, then $f(s', v^i_{\mathtt{a}}) = f(v^i_{\mathtt{a}}, v_{\mathtt{a}}) = 1$ if $i \leq \ell$ and $f(s', v^i_{\mathtt{a}}) = f(v^i_{\mathtt{a}}, v_{\mathtt{a}}) = 0$ otherwise. Thus, the total cost of the edges of the form $(v^i_{\mathtt{a}}, v_{\mathtt{a}})$ is $\sum_{i=1}^{\ell}[(i-1)w_{\varphi_1}] = \frac{1}{2}\ell(\ell-1)w_{\varphi_1}$. By the definition of $f[D]$, there are $\#_{f[D].A}(\mathtt{a})$ edges of the form $(v_{\mathtt{a}}, u_{\mathtt{b}})$ for which $f(v_{\mathtt{a}}, u_{\mathtt{b}}) = 1$. By the definition of a flow, this is also the flow entering the node $v_{\mathtt{a}}$, and we have that $\ell = \#_{f[D].A}(\mathtt{a})$. We conclude that the total cost of the flow on edges of the form $(v^i_{\mathtt{a}}, v_{\mathtt{a}})$ is

$$\sum_{\mathtt{a} \in f[D].A} \left[\frac{1}{2} \cdot \#_{f[D].A}(\mathtt{a}) \cdot (\#_{f[D].A}(\mathtt{a}) - 1) \cdot w_{\varphi_1}\right] = \mathsf{vio}(f[D], \varphi_1) \cdot w_{\varphi_1}\,.$$

The same argument shows that the total cost of the flow on edges of the form $(u_{\mathtt{b}}, u^i_{\mathtt{b}})$ is $\mathsf{vio}(f[D], \varphi_2) \cdot w_{\varphi_2}$.

Finally, the total cost of the edges of the form $(v_{\mathtt{a}}, u_{\mathtt{b}})$ is $\sum_{g \in f[D]}(-w_g)$ by the definition of $f[D]$ and the construction of the network. We conclude that:

$$\mathsf{cost}(f) = \left(\sum_{g \in f[D]} (-w_g)\right) + \mathsf{vio}(f[D], \varphi_1) \cdot w_{\varphi_1} + \mathsf{vio}(f[D], \varphi_2) \cdot w_{\varphi_2}$$

and $\mathsf{cost}(f) = w_D(f[D])$ by definition. ◀

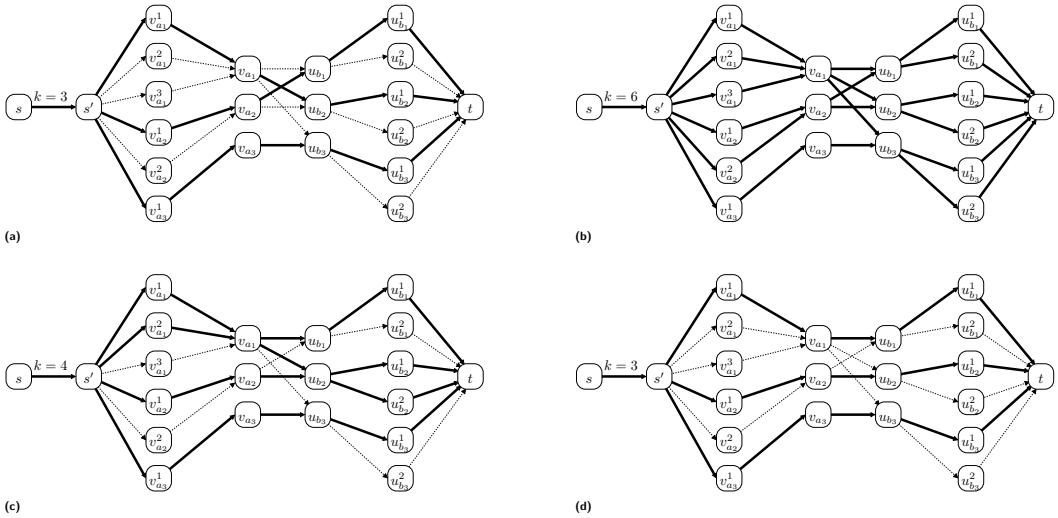


Figure 4 The flow in the network $\mathcal{N}_k$ corresponding to an optimal subset of the database of Figure 2a for different weights.

The next lemma follows straightforwardly from the construction of $\mathcal{N}_k$ and the definition of f_E.

▶ **Lemma 11.** *Every subinstance E of D satisfies* $\mathsf{cost}(f_E) = w_D(E)$.

Now, let E be an optimal subset of D w.r.t. Δ and assume that $|E| = k$. Let f^* be a solution with the minimum cost among all the solutions to MCMF on $\mathcal{N}_1 \ldots, \mathcal{N}_n$. Lemma 11 implies that there is an integral flow f_E in $\mathcal{N}_k$ such that $\mathsf{cost}(f_E) = w_D(E)$. Hence, we have that $\mathsf{cost}(f^*) \leq w_D(E)$. By applying Lemma 10 on f^*, there is another subinstance E' of D such that $w_D(E') = \mathsf{cost}(f^*)$. Since E is an optimal subset, we have that $w_D(E) \leq w_D(E')$. Overall, we have that $\mathsf{cost}(f^*) \leq w_D(E) \leq w_D(E') = \mathsf{cost}(f^*)$, and we conclude that $\mathsf{cost}(f^*) = w_D(E)$. Therefore, by taking the solution with the lowest cost among all solutions to MCMF on $\mathcal{N}_1, \ldots, \mathcal{N}_n$, we indeed find a solution to our problem, and that concludes our proof of Lemma 9.

▶ **Example 12.** Consider again the database of Figure 2a. Assume that:

$$w_{\varphi_1} = w_{\varphi_2} = 2 \qquad w_{f_1} = w_{f_2} = w_{f_3} = w_{f_4} = w_{f_5} = w_{f_6} = 1$$

Since the cost of a violation is "too high" in this case (i.e., it is always cheaper to delete a fact involved in a violation than to keep the violation), an optimal subset in this case is, in fact, an optimal repair in the traditional sense (that is, when the constraints are assumed to be hard constraints). One possible optimal repair in this case is $\{f_2, f_4, f_6\}$. The flow corresponding to this repair in the network $\mathcal{N}_3$ is illustrated in Figure 4a.

Now, assume that:

$$w_{\varphi_1} = w_{\varphi_2} = 1 \qquad w_{f_1} = w_{f_2} = w_{f_3} = w_{f_4} = w_{f_5} = w_{f_6} = 3$$

In this case, the cost of deleting a fact is "too high", since each fact is involved in at most two violations, and the cost of keeping the violation is lower than the cost of removing facts involved in the violation. Therefore, the database itself is an optimal subset, and the corresponding flow in the network $\mathcal{N}_6$ is illustrated in Figure 4b.

As another example, assume that:

$$w_{\varphi_1} = w_{\varphi_2} = 1 \qquad w_{f_1} = w_{f_2} = w_{f_5} = 2, w_{f_3} = w_{f_4} = 1, w_{f_6} = 3$$

Here an optimal subset consists of the facts in $\{f_1, f_2, f_5, f_6\}$, and the corresponding flow in the network $\mathcal{N}_4$ is illustrated in Figure 4c. If we modify the weight of φ_2 and define $w_{\varphi_2} = 4$, while keeping the rest of the weight intact, it is now cheaper to delete the fact f_2 rather than keep the violations it is involved in with f_1 and f_5; hence, an optimal subset in this case is $\{f_1, f_5, f_6\}$, and the corresponding flow in the network $\mathcal{N}_3$ is illustrated in Figure 4d. ⌟

Note that the FD set $\{A \to B\}$ over $R(A, B)$ is in fact a special case of the result of Theorem 14, as we can compute an optimal subset for this FD set using the algorithm described above by defining $w_{B \to A} = 0$. However, this algorithm works only for the case where the single FD is a key and fails to compute the correct solution when the schema contains attributes that do not appear in the FD. The algorithm described in the proof of Theorem 6, on the other hand, can handle this case and does not assume anything about the underlying schema.

5.1 Generalization

We now extend our algorithm beyond bipartite matching to the more general case of a matching constraint. By a "matching constraint" we refer to the case of $\hat{\Delta} = \{X \to Y, X' \to Y'\}$ over a schema $\hat{R}(A_1, \ldots, A_k)$ where $X \cup Y = X' \cup Y' = X \cup X' = \{A_1, \ldots, A_k\}$. An example follows.

▶ **Example 13.** Consider the database of our running example (Figure 1), and the following FDs:

- Flight Airline Date → Origin Destination Airplane,
- Origin Destination Airplane Date → Flight Airline.

The reader can easily verify that these two FDs form a matching constraint. On the other hand, consider the set consisting of the following two FDs:

- Flight Date → Airline Origin Destination Airplane,
- Origin Destination Airplane Date → Flight Airline.

Here, we do not have a matching constraint since while it holds that $X \cup Y = X' \cup Y' = \{\mathsf{Flight}, \mathsf{Airline}, \mathsf{Date}, \mathsf{Origin}, \mathsf{Destination}, \mathsf{Airplane}\}$, the set $X \cup X'$ misses the Airline attribute. ⌟

The generalization of Lemma 9 from $\Delta = \{A \to B, B \to A\}$ over $R(A, B)$ to the general case of a matching constraint is fairly straightforward. Given an input $\hat{D}$ for soft repairing over $\hat{R}$ and $\hat{\Delta}$, we construct an input D over R and Δ by defining unique values $a(\pi_X(\hat{f}))$ and $b(\pi_{X'}(\hat{f}))$ for the projections $\pi_X(\hat{f})$ and $\pi_{X'}(\hat{f})$ over X and X', respectively, of every fact $\hat{f}$ of $\hat{D}$. Then, the database D is simply the set of all the pairs $a(\pi_X \hat{f})$ and $b(\pi_{X'} \hat{f})$ for all facts $\hat{f}$ of D:

$$D \stackrel{\text{def}}{=} \{(a(\pi_X \hat{f}), b(\pi_{X'} \hat{f})) \mid \hat{f} \in \hat{D}\}$$

In addition, we define $w_f \stackrel{\text{def}}{=} w_{\hat{f}}$ whenever $f = (a(\pi_X \hat{f}), b(\pi_{X'} \hat{f}))$ and $w_{A \to B} \stackrel{\text{def}}{=} w_{X \to Y}$ and $w_{B \to A} \stackrel{\text{def}}{=} w_{X' \to Y'}$. Note that the mapping $f \to \hat{f}$ is reversible since $X \cup X' = \{A_1, \ldots, A_k\}$. So, in order to solve soft repairing for $\hat{D}$, we solve it for D and transform every fact f of D into the corresponding fact $\hat{f}$ of $\hat{D}$. We get the following result. The proof is by showing the correctness of the reduction.

▶ **Theorem 14.** *Soft repairing is solvable in polynomial time whenever Δ is a pair of FDs that constitutes a matching constraint.*

Proof. We prove that D has a subset E with $\mathsf{cost}(E \mid D) = k$ if and only if $\hat{D}$ has a subset $\hat{E}$ with $\mathsf{cost}(\hat{E} \mid \hat{D}) = k$. Let E be a subset of D with cost k. Let $\hat{E}$ be a subset of $\hat{D}$ that includes the fact $\hat{f}$ for every $f \in E$. By definition, we have that $\sum_{f \in (D \setminus E) w_f} = \sum_{f \in (\hat{D} \setminus \hat{E}) w_{\hat{f}}}$; hence, it is left to show that $\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)| = \sum_{\hat{\varphi} \in \hat{\Delta}} w_{\hat{\varphi}} |\mathsf{vio}(\hat{E}, \hat{\varphi})|$. Let $f, g \in E$ such that $\{f, g\} \not\models (A \to B)$. Hence, it holds that $f[A] = g[A]$ while $f[B] \neq g[B]$. From the construction of D, we have that $\pi_X \hat{f} = \pi_X \hat{g}$, while $\pi_{X'} \hat{f} \neq \pi_{X'} \hat{g}$. Thus, there is an attribute $A_i \in X'$ such that $\hat{f}[A_i] \neq \hat{g}[A_i]$ and since $A_i \notin X$ and $X \cup Y = \{A_1, \ldots, A_k\}$, it holds that $A_i \in Y$. We conclude that $\{\hat{f}, \hat{g}\} \not\models (X \to Y)$. We can similarly prove that if $\{f, g\} \not\models (B \to A)$, then $\{\hat{f}, \hat{g}\} \not\models (X' \to Y')$. Finally, because $w_{A \to B} = w_{X \to Y}$ and $w_{B \to A} = w_{X' \to Y'}$ it holds that $\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)| = \sum_{\hat{\varphi} \in \hat{\Delta}} w_{\hat{\varphi}} |\mathsf{vio}(\hat{E}, \hat{\varphi})|$.

For the other direction, let $\hat{E}$ be a subset of $\hat{D}$, and let E be the subset of D that includes the fact f for every $\hat{f} \in \hat{E}$. It is again straightforward that $\sum_{f \in (D \setminus E) w_f} = \sum_{f \in (\hat{D} \setminus \hat{E}) w_{\hat{f}}}$. Now, let $\hat{f}, \hat{g} \in \hat{E}$ such that $\{\hat{f}, \hat{g}\} \not\models (X \to Y)$. We have that $\hat{f}[A_i] = \hat{g}[A_i]$ for every $A_i \in X$; thus, $\pi_X \hat{f} = \pi_X \hat{g}$ and from the construction of D, it holds that $f[A] = g[A]$. On the other hand, the fact that $\hat{f}[A_i] \neq \hat{g}[A_i]$ for some $A_i \in Y$ together with the fact that $X \cup Y = X \cup X' = \{A_1, \ldots, A_k\}$ imply that $\pi_{X'} \hat{f} \neq \pi_{X'} \hat{g}$ and $f[B] \neq g[B]$. Hence, $\{f, g\} \not\models (A \to B)$. We can similarly prove that if $\{\hat{f}, \hat{g}\} \not\models (X' \to Y')$, then $\{f, g\} \not\models (B \to A)$, which again implies that $\sum_{\varphi \in \Delta} w_\varphi |\mathsf{vio}(E, \varphi)| = \sum_{\hat{\varphi} \in \hat{\Delta}} w_{\hat{\varphi}} |\mathsf{vio}(\hat{E}, \hat{\varphi})|$, and the concludes our proof. ◀

6 Conclusions and Open Problems

We studied the complexity of soft repairing for functional dependencies, where the goal is to find an optimal subset under penalties of deletion and constraint violation. The problem is harder than that of computing a cardinality repair, and we have developed two new, nontrivial algorithms solving natural special cases. A full classification of the FD sets remains an open challenge for future research; specifically, the question is what fragment of the positive side of the dichotomy of Livshits et al. [14] remains positive when softness is allowed. We have also shown that the problem becomes tractable if we settle for a 3-approximation.

Open Problems

Several directions are left open for future work. A direct open problem is to characterize the class of tractable FDs via a full dichotomy. The simplest sets of FDs where the complexity of soft repairing is open are the following:

- $\{A \to B, A \to C\}$. Note that this problem is different from $\{A \to BC\}$ that consists of a single FD.
- $\{A \to B, B \to A\}$ in the case where the schema has attributes different from A and B, starting with $R(A, B, C)$.
- $\{\emptyset \to A, B \to C\}$.

The problem is also open for classes of constraints that are more general than FDs, including equality-generating dependencies (EGDs), denial constraints, and inclusion dependencies. Yet, the problem for these types of dependencies is open already in the case of cardinality repairs, with the exception of some cases of EGDs [13]. Another clear direction is that of *update repairs* where we are allowed to change cell values instead of (or in addition to) deleting tuples and where complexity results are known for hard constraints [10, 14].

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

2 Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013. URL: `http://www.vldb.org/pvldb/vol6/p1498-papotti.pdf`, `doi:10.14778/2536258.2536262`.

3 Carlo Combi, Matteo Mantovani, Alberto Sabaini, Pietro Sala, Francesco Amaddeo, Ugo Moretti, and Giuseppe Pozzi. Mining approximate temporal functional dependencies with pure temporal grouping in clinical databases. *Comp. in Bio. and Med.*, 62:306–324, 2015. `doi:10.1016/j.compbiomed.2014.08.004`.

4 Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15(3):430–466, August 1990. `doi:10.1287/moor.15.3.430`.

5 Teofilo F. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007. `doi:10.1201/9781420010749`.

6 Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. The most probable database problem. In *BUDA*, 2014. URL: `http://www.sigmod2014.org/buda`.

7 Dorit S Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on computing*, 11(3):555–556, 1982.

8 Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999. `doi:10.1093/comjnl/42.2.100`.

9 Abhay Kumar Jha, Vibhor Rastogi, and Dan Suciu. Query evaluation with soft-key constraints. In *PODS*, pages 119–128, 2008.

10 Solmaz Kolahi and Laks V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, volume 361 of *ACM International Conference Proceeding Series*, pages 53–62. ACM, 2009.

11 Weibang Li, Zhanhuai Li, Qun Chen, Tao Jiang, and Zhilei Yin. Discovering approximate functional dependencies from distributed big data. In *APWeb*, pages 289–301, 2016. `doi:10.1007/978-3-319-45817-5_23`.

12 Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. Approximate denial constraints. *Proc. VLDB Endow.*, 13(10):1682–1695, 2020. URL: `http://www.vldb.org/pvldb/vol13/p1682-livshits.pdf`.

13 Ester Livshits, Ihab F. Ilyas, Benny Kimelfeld, and Sudeepa Roy. Principles of progress indicators for database repairing. *CoRR*, abs/1904.06492, 2019.

14 Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing optimal repairs for functional dependencies. *ACM Trans. Database Syst.*, 45(1):4:1–4:46, 2020. `doi:10.1145/3360904`.

15 Andrei Lopatenko and Leopoldo E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, volume 4353 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2007.

16 Eduardo H. M. Pena, Eduardo Cunha de Almeida, and Felix Naumann. Discovery of approximate (and exact) denial constraints. *Proc. VLDB Endow.*, 13(3):266–278, 2019. `doi:10.14778/3368289.3368293`.

17 Matthew Richardson and Pedro Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, February 2006.

18 Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. A formal framework for probabilistic unclean databases. In *ICDT*, volume 127 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

19 Prithviraj Sen, Amol Deshpande, and Lise Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

Uniform Reliability of Self-Join-Free Conjunctive Queries

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Benny Kimelfeld
Technion - Israel Institute of Technology, Haifa, Israel

Abstract

The *reliability* of a Boolean Conjunctive Query (CQ) over a tuple-independent probabilistic database is the probability that the CQ is satisfied when the tuples of the database are sampled one by one, independently, with their associated probability. For queries without self-joins (repeated relation symbols), the data complexity of this problem is fully characterized in a known dichotomy: reliability can be computed in polynomial time for *hierarchical* queries, and is #P-hard for non-hierarchical queries. Hierarchical queries also characterize the tractability of queries for other tasks: having read-once lineage formulas, supporting insertion/deletion updates to the database in constant time, and having a tractable computation of tuples' Shapley and Banzhaf values.

In this work, we investigate a fundamental counting problem for CQs without self-joins: how many sets of facts from the input database satisfy the query? This is equivalent to the *uniform* case of the query reliability problem, where the probability of every tuple is required to be 1/2. Of course, for hierarchical queries, uniform reliability is in polynomial time, like the reliability problem. However, it is an open question whether being hierarchical is necessary for the uniform reliability problem to be in polynomial time. In fact, the complexity of the problem has been unknown even for the simplest non-hierarchical CQs without self-joins.

We solve this open question by showing that uniform reliability is #P-complete for every non-hierarchical CQ without self-joins. Hence, we establish that being hierarchical also characterizes the tractability of unweighted counting of the satisfying tuple subsets. We also consider the generalization to query reliability where all tuples *of the same relation* have the same probability, and give preliminary results on the complexity of this problem.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory)

Keywords and phrases Hierarchical conjunctive queries, query reliability, tuple-independent database, counting problems, #P-hardness

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.17

Related Version *Full Version*: `https://arxiv.org/abs/1908.07093` [1]

Funding The work of Benny Kimelfeld was supported by the Israel Science Foundation (ISF), Grant 768/19, and the German Research Foundation (DFG) Project 412400621 (DIP program).

Acknowledgements The authors are very grateful to Kuldeep S. Meel and Dan Suciu for insightful discussions on the topic of this paper.

1 Introduction

Probabilistic databases [23] extend the usual model of relational databases by allowing database facts to be uncertain, in order to model noisy and imprecise data. The evaluation of a Boolean query Q over a probabilistic database D is then the task of computing the probability that Q is true under the probability distribution over possible worlds given by D. This computational task has been considered by Grädel, Gurevich and Hirsch [10] as a special case of computing the *reliability* of a query in a model which is nowadays known as

Tuple-Independent probabilistic Databases (TIDs) [6, 23]. In a TID, every fact is associated with a probability of being true, and the truth of every fact is an independent random event. While the TID model is rather weak, query evaluation over TIDs can also be used for probabilistic inference over models with correlations among facts, such as Markov Logic Networks [11, 13]. Hence, studying the complexity of query evaluation on TIDs is the first step towards understanding which forms of probabilistic data can be tractably queried.

To this end, Grädel et al. [10] showed the first Boolean Conjunctive Query (referred to simply as a *CQ* hereafter) for which query evaluation is #P-hard on TIDs. Later, Dalvi and Suciu [6] established a dichotomy on the complexity of evaluating CQs without self-joins (i.e., without repeated relation symbols) over TIDs: if the CQ is *safe* (or *hierarchical* [8, 23] as we explain next), the problem is solvable in polynomial time; otherwise, the problem is #P-hard. (This result was later extended to the class of all CQs and unions of CQs [7].)

The class of *hierarchical* CQs is defined by requiring that, for every two variables x and y, the sets of query atoms that feature x must contain, be contained in, or be disjoint from, the set of atoms that feature y. Interestingly, this class of hierarchical queries was then found to characterize the tractability boundary of other query evaluation tasks for CQs without self-joins, over databases without probabilities (and under conventional complexity assumptions). Olteanu and Huang [18] showed that a query is hierarchical if and only if, for every database, the *lineage* of the query is a read-once formula. Livshits, Bertossi, Kimelfeld and Sebag [15] proved that the hierarchical CQs are precisely the ones that have a tractable *Shapley value* as a measure of responsibility of facts to query answers (a result that was later generalized to CQs with negation [15]); they also conjecture that this complexity classification also holds for another measure of responsibility, namely the *causal effect* [21]. (We discuss these measures again later in this section.) Berkholz, Keppeler and Schweikardt [3] showed that the hierarchical CQs are (up to conventional assumptions of fine-grained complexity) precisely the ones for which we can use an auxiliary data structure to update the query answer in constant time in response to the insertion or deletion of a tuple.

In this paper, we show that the property of being hierarchical also captures the complexity of a fundamental counting problem for CQs without self-joins: *how many sets of facts from the input database satisfy the query?* This problem, which we refer to as *uniform reliability*, is equivalent to query evaluation over a TID where the probability of *every* fact is equal to $\frac{1}{2}$. In particular, it follows from the aforementioned dichotomy that this problem can be solved in polynomial time for every self-join-free hierarchical CQ Q. Yet, if Q is not hierarchical, it does not necessarily mean that Q is intractable already in this uniform setting. Indeed, it was not known whether enforcing uniformity makes query evaluation on TIDs easier, and the complexity of uniform reliability was already open for the simplest case of a non-hierarchical CQ: $Q_1 \coloneq \mathsf{R}(x), \mathsf{S}(x, y), \mathsf{T}(y)$. The proofs of #P-hardness of Dalvi and Suciu [6] require TIDs with deterministic facts (probability 1), in addition to $\frac{1}{2}$, already in the case of Q_1. Here, we address this problem and show that the dichotomy is also true for the uniform reliability problem. In particular, uniform reliability is #P-complete for every non-hierarchical CQ without self-joins (and solvable in polynomial time for every hierarchical CQ without self-joins).

The uniform reliability problem that we study is a basic combinatorial problem on CQs, and a natural restricted case of query answering on TIDs, but it also has a direct application for quantifying the impact (or responsibility) of a fact f on the result of a CQ Q over ordinary (non-probabilistic) databases. One notion of tuple impact is the aforementioned causal effect, defined as the difference between two quantities: the probability of Q conditioning on the existence of f, minus the probability of Q conditioning on the absence of f [21]. This causal

effect was recently shown [3] to be the same as the *Banzhaf power index*, studied in the context of wealth distribution in cooperative game theory [9] and applied, for instance, to voting in the New York State Courts [12]. One notion of causal effect (with so-called *endogenous* facts) is defined by viewing the ordinary database as a TID where the probability of every fact is $\frac{1}{2}$. Therefore, computing the causal effect amounts to solving two variations of uniform reliability, corresponding to the two quantities. In fact, it is easy to see that all of our results apply to each of these two variations.

Uniform reliability also relates to the aforementioned computation of a tuple's Shapley value, a measure of wealth distribution in cooperative game theory that has been applied to many use cases [20, 22]. Livshits et al. [15] showed that computing a tuple's Shapley value can be reduced to a generalized variant of uniform reliability. Specifically, for CQs, computing the Shapley value (again for *endogenous* facts) amounts to calculating the number of subinstances that satisfy Q *and* have precisely m tuples (for a given number m). This generalization of uniform reliability is tractable for every hierarchical CQ without self-joins [15]. Clearly, our results here imply that this generalization is intractable for every non-hierarchical CQ without self-joins, allowing us to conclude that the complexity dichotomy also applies to this generalization.

Our investigation can be viewed as a first step towards the study of problems that lie in between uniform reliability and probabilistic query answering over TIDs. For instance, a natural variant is the one where the probability of each tuple of the database is the same, but not necessarily $\frac{1}{2}$. This problem can arise, for example, in scenarios of network reliability, where all connections are equally important and have the same independent probability of failure. A more general case is the one where the probabilities for every relation are the same, but different relations may be associated with different probabilities. This corresponds to data integration scenarios where every relation is a resource with a different level of trust (e.g., enterprise data vs. Web data vs. noisy sensor data). In this paper, we formalize this generalization and ask which combinations of CQs and probability assignments make the problem intractable. We do not completely answer this question, but propose preliminary results for the query Q_1 mentioned earlier: we show that some combinations of probabilities can be easily proved hard using our main result, while others can be proved hard with other techniques.

Related work. As explained earlier, our work is closely related to existing literature on query evaluation over probabilistic databases. The dichotomy of Dalvi and Suciu [6] for CQs without self-joins requires tuples with probabilities $\frac{1}{2}$ and 1. This is also the case for their generalized dichotomy on CQs without self-joins where some relations can be required to be deterministic (i.e., all tuples have probability 1) while tuples in the remaining relations can have arbitrary probabilities (including 1). The later generalization of the dichotomy by Dalvi and Suciu [7] to CQs with self-joins and to UCQs required an unbounded class of probabilities, not just $\frac{1}{2}$ and 1. In very recent work, Kenig and Suciu [14] have strengthened the generalized dichotomy and showed that probabilities $\frac{1}{2}$ and 1 suffice for UCQs as well.[1] In that work, they also investigate uniform reliability (that we study here, i.e., where $\frac{1}{2}$ is the only nonzero probability allowed) and prove #P-hardness for the so-called unsafe "final type-I" queries. As they explain in their discussion on the work of this paper (which was posted as a preprint before theirs), their result on uniform reliability complements ours, and it is not clear if any of these two results can be used to prove the other.

[1] Kenig and Suciu refer to this case as TID with probabilities from $\{0, \frac{1}{2}, 1\}$; we mean the same thing, as in this paper we assume that tuples with probability zero are simply ignored.

The work of this paper also relates to rewriting techniques used in the case of DNF formulas to reduce weighted model counting to unweighted model counting [5]. Nevertheless, the results and techniques for this problem are not directly applicable to ours, since model counting for CQs translates to DNFs of a very specific shape (namely, those that can be obtained as the lineage of the query).

Another superficially related problem is that of *symmetric model counting* [2]. This is a variant of uniform reliability where each relation consists of *all possible tuples* over the corresponding domain, and so each fact carries the same weight: these assumptions are often helpful to make model counting tractable. The assumption that we make is much weaker: we do not deal with symmetric databases, but rather with arbitrary databases where all facts of the database (but *not* necessarily all possible facts over the domain) have the same uniform probability of $\frac{1}{2}$. For this reason, the tractability results of Beame et al. [2] do not carry over to our setting. In terms of hardness results, [2, Theorem 3.1] shows the $\#\mathrm{P}_1$-hardness of symmetric model counting (hence of uniform reliability) for a specific FO^3 sentence, and [2, Corollary 3.2] shows a $\#\mathrm{P}_1$-hardness result for *weighted* symmetric model counting for a specific CQ (without assuming self-join-freeness). Hence, these results do not determine the complexity of uniform reliability for self-join-free CQs as we do here.

There is a closer connection to existing dichotomy results on counting *database repairs* [16, 17]. In this setting, the input database may violate the primary key constraints of the relations, and a repair is obtained by selecting one fact from every collection of conflicting facts (i.e., distinct facts that agree on the key): the *repair counting problem* asks how many such repairs satisfy a given CQ. In particular, it can easily be shown that for a CQ Q, there is a reduction from the uniform reliability of Q to repair counting of another CQ Q'. Yet, this reduction can only explain cases of tractability (namely, where Q is hierarchical) which, as explained earlier, are already known. We do not see how to design a reduction in the other direction, from repair counting to uniform reliability, in order to show our hardness result.

Finally, our work relates to the study of the Constraint Satisfaction Problem (CSP). However, there are two key differences. First, we study query evaluation in terms of homomorphisms *from* a fixed CQ, whereas the standard CSP phrasing talks about homomorphisms *to* a given template. Second, the standard counting variant of CSP (namely, #CSP), for which Bulatov has proved a dichotomy [4], is about counting *the number of homomorphisms*, whereas we count *the number of subinstances* for which a homomorphism exists. For these reasons, it is not clear how results on CSP and #CSP can be helpful towards our main result.

Organization. We give preliminaries in Section 2. In Section 3, we formally state the studied problem and main result, that is, the dichotomy on the complexity of uniform reliability for CQs without self-joins. We prove this result in Sections 4–6. We discuss a generalization to arbitrary uniform probabilities in Section 7, and conclude in Section 8. Missing proofs can be found in the full version of this paper [1].

2 Preliminaries

We begin with some preliminary definitions and notation that we use throughout the paper. We first define databases and conjunctive queries, before introducing the task of probabilistic query evaluation, and the uniform reliability problem that we study.

Databases. A (relational) *schema* $\mathbf{S}$ is a collection of *relation symbols* with each relation symbol ρ in $\mathbf{S}$ having an associated arity. We assume a countably infinite set Const of *constants* that are used as database values. A *fact* over $\mathbf{S}$ is an expression of the form

$\rho(c_1, \ldots, c_k)$ where ρ is a relation symbol of $\mathbf{S}$, where k is the arity of ρ, and where $c_1, \ldots, c_k$ are values of Const. An *instance* I over $\mathbf{S}$ is a finite set of facts. In particular, we say that an instance J is a *subinstance* of an instance I if we have $J \subseteq I$.

Conjunctive queries. This paper focuses on queries in the form of a Boolean Conjunctive Query, which we refer to simply as a *CQ*. Intuitively, a CQ Q over the schema $\mathbf{S}$ is a relational query definable as an existentially quantified conjunction of atoms. Formally, a CQ is a first-order formula of the form $Q \text{ :- } \rho_1(\vec{\tau}_1), \ldots, \rho_n(\vec{\tau}_m)$ where each $\rho_i(\vec{\tau}_i)$ is an *atom* of Q, formed of a relation symbol of $\mathbf{S}$ and of a tuple $\vec{\tau}_i$ of constants and (existentially quantified) variables, with the same arity as ρ_i. In the context of a CQ Q, we omit the schema $\mathbf{S}$ and implicitly assume that $\mathbf{S}$ consists of the relation symbols that occur in Q (with the arities that they have in Q); in that case, we may also refer to an instance I over $\mathbf{S}$ as an instance *over* Q. We write $I \models Q$ to state that the instance I satisfies Q. We denote by the set of all subinstances J of I that satisfy Q by:

$$\mathrm{Mod}(Q, I) := \{J \subseteq I \mid J \models Q\}.$$

A *self-join* in a CQ Q is a pair of distinct atoms over the same relation symbol. For example, in $Q \text{ :- } \mathsf{R}(x, y), \mathsf{S}(x), \mathsf{R}(y, z)$, the first and third atoms constitute a self-join. Our analysis in this paper is restricted to CQs *without self-joins*, that we also call *self-join-free*.

Let Q be a CQ. For each variable x of Q, we denote by $\mathrm{atoms}(x)$ the set of atoms $\rho_i(\vec{\tau}_i)$ of Q where x occurs. We say that Q is *hierarchical* [6] if for all variables x and x' one of the following three relations hold: $\mathrm{atoms}(x) \subseteq \mathrm{atoms}(x')$, $\mathrm{atoms}(x') \subseteq \mathrm{atoms}(x)$, or $\mathrm{atoms}(x) \cap \mathrm{atoms}(x') = \emptyset$. The simplest non-hierarchical self-join-free CQ is Q_1, which we already mentioned in the introduction:

$$Q_1 \text{ :- } \mathsf{R}(x), \mathsf{S}(x, y), \mathsf{T}(y) \tag{1}$$

Probabilistic query evaluation. The problem of *probabilistic query evaluation* over tuple-independent databases [23] is defined as follows.

▶ **Definition 2.1.** *The problem of* probabilistic query evaluation *(or PQE) for a CQ Q, denoted $\mathsf{PQE}(Q)$, is that of computing, given an instance I over Q and an assignment $\pi : I \to [0, 1]$ of a probability $\pi(f)$ to every fact f, the probability that Q is true, namely:*

$$\Pr(Q, I, \pi) := \sum_{J \in \mathrm{Mod}(Q,I)} \prod_{f \in J} \pi(f) \times \prod_{f \in I \setminus J} (1 - \pi(f)) .$$

We again study the *data complexity* of this problem, and we assume that the probabilities attached to the instance I are rational numbers represented by their integer numerator and denominator.

PQE was first studied by Grädel, Gurevich and Hirsch [10] as *query reliability* (which they also generalize beyond Boolean queries). They identified a Boolean CQ Q with self-joins such that the reliability of Q is #P-hard to compute. Dalvi and Suciu [6, 7] then studied the PQE problem, culminating in their dichotomy for the complexity of PQE on unions of conjunctive queries with self-joins [7]. In this paper, we only consider their earlier study of CQs without self-joins [6]. They characterize, under conventional complexity assumptions, the self-join-free CQs where PQE is solvable in PTIME. They state the result in terms of safe query plans ("safe CQs"), but the term "hierarchical" was adopted in later publications [8, 23]:

▶ **Theorem 2.2.** [6] *Let Q be a CQ without self-joins. If Q is hierarchical, then $\mathsf{PQE}(Q)$ is solvable in polynomial time. Otherwise, $\mathsf{PQE}(Q)$ is #P-hard.*

Recall that #P is the complexity class of problems that count witnesses of an NP-relation (e.g., satisfying assignments of a logical formula, vertex covers of a graph, etc.). A function F is #P-hard if every function in #P has a polynomial-time *Turing reduction* (or *Cook reduction*) to F.

We stress that Theorem 2.2 applies to CQs *without* self-joins. In the presence of self-joins, being hierarchical is still necessary for tractability, but no longer sufficient [23, Theorem 4.23, Proposition 4.25].

Uniform reliability. We study the query reliability problem (which we equivalently refer to as PQE), and focus on the uniform variant of this problem, where the probability of every fact is $\frac{1}{2}$. Equivalently, the task is to count the subinstances that satisfy the query (up to division/multiplication by 2^n where n is the number of facts in the instance). Formally:

▶ **Definition 2.3.** *The problem of* uniform reliability *for a CQ Q, denoted $\mathsf{UR}(Q)$, is that of determining, given an instance I over Q, how many subinstances of I satisfy Q. In other words, $\mathsf{UR}(Q)$ is the problem of computing $|\mathrm{Mod}(Q, I)|$ given I. We study the* data complexity *of this problem, i.e., Q is fixed and the complexity is a function of the input I.*

3 Problem Statement and Main Result

Let Q be a CQ without self-joins. It follows from Theorem 2.2 that, if Q is hierarchical, then $\mathsf{UR}(Q)$ is solvable in polynomial time. Indeed, there is a straightforward reduction from $\mathsf{UR}(Q)$ to $\mathsf{PQE}(Q)$: given an instance I for Q, let $\pi : I \to [0, 1]$ be the function that assigns to every fact of I the probability $\pi(f) = \frac{1}{2}$. Then we have:

$$|\mathrm{Mod}(Q, I)| = 2^{|I|} \times \Pr(Q, I, \pi)$$

because every subset of Q has the same probability, namely $2^{-|I|}$.

However, the other direction is not evident. If Q is non-hierarchical, we know that $\mathsf{PQE}(Q)$ is #P-hard, but we do not know whether the same is true of $\mathsf{UR}(Q)$. Indeed, this does not follow from Theorem 2.2 (as uniform reliability is a restriction of PQE), and it does not follow from the proof of the theorem either. Specifically, the reduction that Dalvi and Suciu [6] used to show hardness consists of two steps.

1. Proving that $\mathsf{PQE}(Q_1)$ is #P-hard (where Q_1 is defined in (1)).
2. Constructing a polynomial-time Turing reduction from $\mathsf{PQE}(Q_1)$ to $\mathsf{PQE}(Q)$ for every non-hierarchical CQ Q without self-joins.

In both steps, the constructed instances I consist of facts with two probabilities: $\frac{1}{2}$ and 1 (i.e., *deterministic facts*). If all facts had probability $\frac{1}{2}$, then we would get a reduction to our $\mathsf{UR}(Q)$ problem. However, the proof crucially relies on deterministic facts, and we do not see how to modify it to give the probability $\frac{1}{2}$ to all facts. This is true for both steps. Even for the first step, the complexity of $\mathsf{UR}(Q_1)$ has been unknown so far. For the second step, it is not at all clear how to reduce from $\mathsf{UR}(Q_1)$ to $\mathsf{UR}(Q)$, even if $\mathsf{UR}(Q_1)$ is proved to be #P-hard.

In this paper, we resolve the question and prove that $\mathsf{UR}(Q)$ is #P-complete whenever Q is a non-hierarchical CQ without self-joins. Hence, we establish that the dichotomy of Theorem 2.2 also holds for uniform reliability. Our main result is:

▶ **Theorem 3.1.** *Let Q be a CQ without self-joins. If Q is hierarchical, then $\mathsf{UR}(Q)$ is solvable in polynomial time. Otherwise, $\mathsf{UR}(Q)$ is #P-complete.*

As explained earlier, the tractability of $\mathsf{UR}(Q)$ for hierarchical queries follows from Theorem 2.2. (Note that the theorem assumes the self-join freeness of the query.) Membership of $\mathsf{UR}(Q)$ in #P is straightforward. Thus, our technical contribution is the following:

▶ **Theorem 3.2.** *Let Q be a non-hierarchical CQ without self-joins. Then $\mathsf{UR}(Q)$ is #P-hard.*

A preliminary observation is that this claim follows from the hardness of a specific family of non-hierarchical self-join-free CQs. We call them the *$Q_{r,s,t}$-queries*, and they have the following form for some natural numbers $r, s, t > 0$:

$$Q_{r,s,t} \text{ :- } \mathsf{R}_1(x), \ldots, \mathsf{R}_r(x), \mathsf{S}_1(x,y), \ldots, \mathsf{S}_s(x,y), \mathsf{T}_1(y), \ldots, \mathsf{T}_t(y) \tag{2}$$

These queries always have two variables and no constants, and relations of arity 1 or 2. Note that $Q_{1,1,1}$ is the same as Q_1 (introduced as Equation (1) on page 5). We can show that, for any non-hierarchical self-join-free CQ Q, there is a reduction to the $\mathsf{UR}(Q)$ problem from the problem $\mathsf{UR}(Q_{r,s,t})$ for some $r, s, t > 0$. Formally:

▶ **Proposition 3.3.** *Let Q be a non-hierarchical CQ without self-joins. We can compute natural numbers $r, s, t > 0$ such that there is a polynomial-time Turing reduction from $\mathsf{UR}(Q_{r,s,t})$ to $\mathsf{UR}(Q)$.*

Proof sketch. We only sketch the case where Q has no constants, as the general case is similar. As Q is non-hierarchical, we can find two variables x and y such that the sets $\mathrm{atoms}(x)$ and $\mathrm{atoms}(y)$ intersect and are incomparable. We take $r := |\mathrm{atoms}(x) \setminus \mathrm{atoms}(y)|$, $s := |\mathrm{atoms}(x) \cap \mathrm{atoms}(y)|$, and $t := |\mathrm{atoms}(y) \setminus \mathrm{atoms}(x)|$. We then reduce from an instance of $\mathsf{UR}(Q_{r,s,t})$ to an instance of $\mathsf{UR}(Q)$.

To do so, given an input instance I, we rewrite the S_i-facts $\mathsf{S}_i(a,b)$ for $1 \leq i \leq s$ by facts of the i-th relation of $\mathrm{atoms}(x) \cap \mathrm{atoms}(y)$, by reusing a and b at the positions where x and y are respectively used in Q for that relation, and filling the other positions with some fixed constant c. We rewrite the R_i-facts and the T_i-facts in the same manner to facts corresponding to the relations of $\mathrm{atoms}(x) \setminus \mathrm{atoms}(y)$ and to relations of $\mathrm{atoms}(y) \setminus \mathrm{atoms}(x)$ respectively, using the same fixed constant c for positions of the new facts where neither x nor y was used in Q. Last, for every relation of Q which is not used in $\mathrm{atoms}(x) \cup \mathrm{atoms}(y)$, we create one fact in the instance where all positions are filled with the fixed constant c. This rewriting is in polynomial time.

We can then show that UR for $Q_{r,s,t}$ on I reduces to UR for Q on the rewritten instance, which establishes the result. Note that the correctness of this process relies on the self-join-freeness of Q. ◀

From Proposition 3.3 we conclude that it suffices to prove hardness for the $Q_{r,s,t}$-queries:

▶ **Theorem 3.4.** *For all $r, s, t > 0$, the problem $\mathsf{UR}(Q_{r,s,t})$ is #P-hard.*

This implies in particular that $\mathsf{UR}(Q_1)$ is #P-hard. We prove this theorem in Sections 4–6, and then investigate a generalization in Section 7.

4 Defining the Main Reduction

In this section and the two next ones, we show the #P-hardness of $\mathsf{UR}(Q_{r,s,t})$ (Theorem 3.4). Fix the arbitrary values $r, s, t > 0$ from the theorem statement. We reduce from the #P-hard problem of counting the number of independent sets of a bipartite graph. The input to this problem is a bipartite graph $G = (R \cup T, S)$ where $S \subseteq R \times T$, and the goal is to calculate

the number P of *independent-set pairs* (R', T') with $R' \subseteq R$ and $T' \subseteq T$, that is, pairs such that $R' \times T'$ is disjoint from S. This problem is the same as computing the number of falsifying assignments of a so-called *monotone partitioned 2-DNF formula*, i.e., a monotone Boolean formula in disjunctive normal form over variables from two disjoint sets $\mathcal{X}$ and $\mathcal{Y}$ where every clause is the conjunction of one variable of $\mathcal{X}$ and one variable of $\mathcal{Y}$. Counting the satisfying assignments of such formulas is #P-hard [19], so it is also #P-hard to count falsifying assignments, and thus to count independent-set pairs.

Let us fix $G = (R \cup T, S)$ as the input to the problem. Our proof consists of three parts. First, in the present section, we introduce several gadgets and use them to build the various instances of $\mathsf{UR}(Q_{r,s,t})$ to which we reduce. Second, in Section 5, we explain how to obtain a linear equation system that connects the number P of independent-set pairs of G to the results of $\mathsf{UR}(Q_{r,s,t})$ on our instances. Last, in Section 6, we argue that the matrix of this system is invertible, so we can recover P and conclude the reduction, showing Theorem 3.4.

Defining the Gadgets. We define the gadgets that we will use in the reduction as building blocks for our instances of $\mathsf{UR}(Q_{r,s,t})$. Recall that R_i, S_i, and T_i are the relations that occur in $Q_{r,s,t}$ (see Equation (2)). For all i, we collectively refer to a fact over R_i, S_i and T_i as an R_*-fact, an S_*-fact, and a T_*-fact, respectively. In our reduction, we will use multiple copies of the gadgets, instantiated with specific elements that will intuitively serve as endpoints to the gadgets. There are two types of gadgets:

- The *(a,b)-gadget* is an instance with two elements a and b (which are intuitively the endpoints), and the following facts (noting that they satisfy the query):

$$\mathsf{R}_1(a), \ldots, \mathsf{R}_r(a), \qquad \mathsf{S}_1(a,b), \ldots, \mathsf{S}_s(a,b), \qquad \mathsf{T}_1(b), \ldots, \mathsf{T}_t(b)$$

We will need to count the possible worlds of this gadget and of subsequent gadgets, because these quantities will be important in the reduction to understand the link between the independent-set pairs of G and the subinstances of our instances of $\mathsf{UR}(Q_{r,s,t})$ that satisfy the query. To this end, we denote by λ_{R} the number of possible worlds of the (a,b)-gadget that violate $Q_{r,s,t}$ when we fix the R_*-facts on a to be present. We easily compute: $\lambda_{\mathsf{R}} = 2^{s+t} - 1$. Similarly, we denote by λ_{T} the number of possible worlds that violate $Q_{r,s,t}$ when we fix the T_*-facts on b to be present. We have: $\lambda_{\mathsf{T}} = 2^{s+r} - 1$. Last, we denote by $\bar{\lambda}_{\mathsf{T}} = 2^{s+r}$ the number of possible worlds when we fix the T_*-facts to be absent (all these possible worlds violate $Q_{r,s,t}$), and denote by $\bar{\lambda}_{\mathsf{R}} = 2^{s+t}$ the number of possible worlds when we fix the R_*-facts to be absent (again, all violate $Q_{r,s,t}$).

- The *(a,b,c,d)-gadget* is an instance with elements a, b, c, and d, and the following facts:

$$\mathsf{R}_1(a), \ldots, \mathsf{R}_r(a), \quad \mathsf{S}_1(a,b), \ldots, \mathsf{S}_s(a,b), \quad \mathsf{T}_1(b), \ldots, \mathsf{T}_t(b), \quad \mathsf{S}_1(c,b), \ldots, \mathsf{S}_s(c,b),$$
$$\mathsf{R}_1(c), \ldots, \mathsf{R}_r(c), \quad \mathsf{S}_1(c,d), \ldots, \mathsf{S}_s(c,d), \quad \mathsf{T}_1(d), \ldots, \mathsf{T}_t(d).$$

We illustrate the gadget below, where every vertex represents a domain element, every edge represents a pair of elements occurring in a fact, and unary and binary facts are simply written as relation names, respectively above their element and above their edge:

$$a \xrightarrow{\mathsf{R}_1, \ldots, \mathsf{R}_r \quad \mathsf{S}_1, \ldots, \mathsf{S}_s} b \xleftarrow{\mathsf{T}_1, \ldots, \mathsf{T}_t \quad \mathsf{S}_1, \ldots, \mathsf{S}_s} c \xrightarrow{\mathsf{R}_1, \ldots, \mathsf{R}_r \quad \mathsf{S}_1, \ldots, \mathsf{S}_s} d \xleftarrow{\mathsf{T}_1, \ldots, \mathsf{T}_t}$$

We denote by γ its number of possible worlds that violate $Q_{r,s,t}$ where we fix the R_*-facts on a and the T_*-facts on d to be present. We denote by δ_{R} its number of possible worlds that violate $Q_{r,s,t}$ when we fix the R_*-facts on a to be present and the T_*-facts on d to be absent,

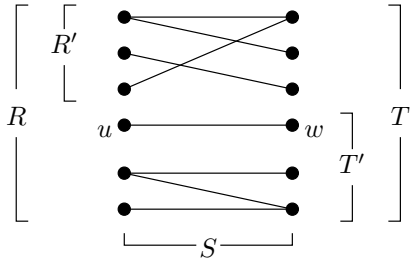


Figure 1 Example of the bipartite graph $G = (R \cup T, S)$ and an independent set (R', T').

and symmetrically denote by δ_{T} its number of possible worlds that violate $Q_{r,s,t}$ when fixing the R_*-facts on a to be absent and the T_*-facts on d to be present. Last, we denote by $\delta_\bot$ its number of possible worlds that violate $Q_{r,s,t}$ when we fix the R_*-facts on a and the T_*-facts on d to be absent. We will study the quantities γ, δ_{R}, δ_{T}, $\delta_\bot$ in two lemmas in Section 6.

Defining the Reduction. Having defined the various gadgets that we will use, let us describe the instances that we construct from our input bipartite graph $G = (R \cup T, S)$ (see Figure 1 for an illustration of the graph). The *vertices* of G are the elements of R and T, and its *edges* are the pairs in S.

Let us now write $m := |S|$, and define the following large (but polynomial) values. We will use them as parameters when building the various UR instances to which we will reduce.

$$M_1 := 4ms + 1$$
$$M_2 := M_1 + 2m(t+s)M_1 + 1$$
$$M_3 := M_1 + M_2 + |R|\,(t+s)M_2 + 1$$

Fix $M := (|R|+1) \times (|T|+1) \times (m+1)^3$, the number of instances to which we will reduce. Now, for each $0 \leq p < M$, we construct the instance D_p on the schema $Q_{r,s,t}$, featuring:

- One element u for each vertex $u \in R$ of G, with all the facts $\mathsf{R}_1(u), \ldots, \mathsf{R}_r(u)$
- One element w for each vertex $w \in T$ of G, with all the facts $\mathsf{T}_1(w), \ldots, \mathsf{T}_t(w)$
- For every edge $(u, w) \in S$ of G, create:
  - p copies of the $(u, *, *, w)$-gadget connecting u and w (using fresh elements for b and c in each copy, as denoted by the $*$'s).
  - $M_1 \times p$ copies of the $(u, *)$-gadget (using a fresh element for b in each copy).
- For each element $u \in R$, create $M_2 \times p$ copies of the $(u, *)$-gadget.
- For each element $w \in T$, create $M_3 \times p$ copies of the $(*, w)$-gadget.

It is clear that this construction is in polynomial time in the input G for each $0 \leq p < M$, because the values M_1, M_2, M_3 are polynomial, so the construction is in polynomial-time overall because M is polynomial. Observe that the construction of D_p is designed to ensure that any match of the query $Q_{r,s,t}$ on a possible world of D_p will always be contained in the facts of one of the gadgets (plus the facts on the elements u and w of the first two bullet points). This means that we can determine if the query is true in the possible worlds simply by looking separately at the facts of each gadget (and at the facts on the u and w).

Now, coming back to our reduction, for each $0 \leq p < M$ we denote by N_p the number of subinstances of D_p that violate $Q_{r,s,t}$. Each of these values can be computed in polynomial time using our oracle for $\mathsf{UR}(Q_{r,s,t})$: for $0 \leq p < M$, we build D_p, call the oracle to obtain the number $|\mathrm{Mod}(Q_{r,s,t}, D_p)|$ of subinstances that satisfy $Q_{r,s,t}$, and compute: $N_p := 2^{|D_p|} - |\mathrm{Mod}(Q_{r,s,t}, D_p)|$.

Hence, in our reduction, given the input bipartite graph G, we have constructed the instances D_p and used our oracle to compute the number N_p of subinstances of each D_p that violate $Q_{r,s,t}$, for each $0 \leq p < M$, and this process is in PTIME. In the next section, we explain how we can use a linear equation system to recover from the numbers N_p the answer to our original problem on $G = (R \cup T, S)$, i.e., the number P of independent-set pairs of G.

5 Obtaining the Equation System

To define the linear equation system, it will be helpful to introduce some parameters about subsets of vertices of the bipartite graph. For any $R' \subseteq R$ and $T' \subseteq T$, we write the following:

- $\mathrm{c}(R', T')$ to denote the number of edges of S that are *contained* in $R' \times T'$, that is, they have both endpoints in $R' \cup T'$. Formally,

$$\mathrm{c}(R', T') := |(R' \times T') \cap S|\,.$$

- $\mathrm{d}(R', T')$ to denote the number of edges of S that are *dangling from* R', that is, they have one endpoint in R' and the other in $T \setminus T'$. Formally,

$$\mathrm{d}(R', T') := |(R' \times (T \setminus T')) \cap S|\,.$$

- $\mathrm{d}'(R', T')$ to denote the number of edges of S that are *dangling from* T', that is, they have one endpoint in $R \setminus R'$ and the other in T'. Formally,

$$\mathrm{d}'(R', T') := |((R \setminus R') \times T') \cap S|\,.$$

- $\mathrm{e}(R', T')$ to denote the number of edges of S that are *excluded* from $R' \cup T'$, that is, they have no endpoint in $R' \cup T'$. Formally,

$$\mathrm{e}(R', T') := |S \setminus (R' \times T')|\,.$$

It is immediate by definition that, for any R' and T', every edge of S is either contained in $R' \times T'$, dangling from R', dangling from T', or excluded from $R' \cup T'$. Hence, we clearly have

$$\mathrm{c}(R', T') + \mathrm{d}(R', T') + \mathrm{d}'(R', T') + \mathrm{e}(R', T') = m.$$

Observe that a pair (R', T') is an independent-set pair of G iff $\mathrm{c}(R', T') = 0$. Thus, given the input G to the reduction, our goal is to compute the following quantity:

$$P = |\{(R', T') \mid R' \subseteq R,\ T' \subseteq T,\ \mathrm{c}(R', T') = 0\}| = \sum_{R' \subseteq R,\ T' \subseteq T,\ \mathrm{c}(R', T') = 0} 1 \tag{3}$$

Let us define the variables on the input graph G that we will use to express P, and that we will recover from the values N_p.

Picking Variables. Our goal is to construct a linear equation system relating the quantity that we wish to compute, namely P, and the quantities provided by our oracle, namely N_p for $0 \leq p < M$. Instead of using P directly, we will construct a system connecting N_p to quantities on G that we now define, from which we will be able to recover P. We call these quantities *variables* because they are unknown and our goal in the reduction is to compute them from the N_p to recover P.

Let us introduce, for each $0 \leq i \leq |R|$, for each $0 \leq j \leq |T|$, for each $0 \leq c, d, d' \leq m$, the variable $X_{i,j,c,d,d'}$, that stands for the number of pairs (R', T') with $|R'| = i$, with $|T'| = j$, and with c- and d- and d′-values exactly as indicated. (We do not need e as a parameter here because it is determined from c, d, d′.) Formally:

$$X_{i,j,c,d,d'} := |\{(R', T') \mid R' \subseteq R\,,\, T' \subseteq T\,,\, |R'| = i\,,\, |T'| = j,\\ \mathrm{c}(R', T') = c\,,\, \mathrm{d}(R', T') = d\,,\, \mathrm{d}'(R', T') = d'\}|$$

For technical reasons, let us define, for all i, j, c, d, d', other variables, which are the ones that we will actually use in the equation system:

$$Y_{i,j,c,d,d'} := (2^r - 1)^{|R|-i} \times (2^t - 1)^{|T|-j} \times X_{i,j,c,d,d'}$$

Getting our Answer from the Variables. Let us now explain why we can compute our desired value P (the number of independent-set pairs of G) from the variables $Y_{i,j,c,d,d'}$. Refer back to Equation (3), and let us split this sum according to the values of the parameters $i = |R'|$, $j = |T'|$, and $\mathrm{d}(R', T')$, $\mathrm{d}'(R', T')$. Using our variables $X_{i,j,c,d,d'}$, this gives:

$$P = \sum_{0 \leq i \leq |R|} \sum_{0 \leq j \leq |T|} \sum_{0 \leq d, d' \leq m} X_{i,j,0,d,d'}$$

We can insert the variables $Y_{i,j,c,d,d'}$ instead of $X_{i,j,c,d,d'}$ in the above, obtaining:

$$P = \sum_{0 \leq i \leq |R|} \sum_{0 \leq j \leq |T|} \sum_{0 \leq d, d' \leq m} \frac{Y_{i,j,0,d,d'}}{(2^r - 1)^{|R|-i} \times (2^t - 1)^{|T|-j}} \tag{4}$$

This equation justifies that, to compute the quantity P that we are interested in, it suffices to compute the value of the variables $Y_{i,j,0,d,d'}$ for all $0 \leq i \leq |R|$, $0 \leq j \leq |T|$, and $0 \leq d, d' \leq m$. If we can compute all these quantities in polynomial time, then we can use the equation above to compute P in polynomial time, completing the reduction.

Designing the Equation System. We will now design a linear equation system that connects the quantities N_p for $0 \leq p < M$ computed by our oracle to the quantities $Y_{i,j,c,d,d'}$ for all $0 \leq i \leq |R|$, $0 \leq j \leq |T|$, $0 \leq c, d, d' \leq m$ that we wish to compute. To do so, write the vector $\vec{N} = (N_0, \ldots, N_{M-1})$, and the vector $\vec{Y} = (Y_{0,0,0,0,0}, \ldots, Y_{|R|,|T|,m,m,m})$. We will describe an M-by-M matrix A so that we have the equation $\vec{N} = A\vec{Y}$. We will later justify that the matrix A is invertible, so that we can compute $\vec{Y}$ from $\vec{N}$ and conclude the proof.

To define the matrix A, let us consider arbitrary subsets $R' \subseteq R$ and $T' \subseteq T$, and an arbitrary $0 \leq p < M$, and let us denote by $\mathcal{D}_p(R', T')$ the set of subinstances of D_p where the set of vertices of R on which we have kept *all* R_*-facts is precisely R', and where the set of vertices on which we have kept *all* T_*-facts is precisely T'. In other words, an instance $I' \subseteq D_p$ is in $\mathcal{D}_p(R', T')$ if *(a)* I' contains all R_*-facts on elements of R' and all T_*-facts on elements of T', *and (b)* for each vertex in $R \setminus R'$, there is at least one R_*-fact missing from I', and for each vertex in $T \setminus T'$, there is at least one T_*-fact missing from I'. It is clear that the $\mathcal{D}_p(R', T')$ form a partition of the subinstances of D_p, so that:

$$N_p = \sum_{R' \subseteq R, T' \subseteq T} |\{I' \in \mathcal{D}_p(R', T') \mid I' \not\models Q_{r,s,t}\}| \tag{5}$$

Let us now study the number in the above sum for each R' and T', that is, the number of instances in $\mathcal{D}_p(R', T')$ that violate the query. We can show the following by performing some accounting over all gadgets in the construction.

▷ Claim 5.1. For any $0 \leq p < M$, for any choice of R' and T', writing $i := |R'|$, $j := |T'|$, $c := \mathrm{c}(R', T')$, $d := \mathrm{d}(R', T')$, $d' := \mathrm{d}'(R', T')$, $e := \mathrm{e}(R', T') = m - c - d - d'$, we have:

$$|\{I' \in \mathcal{D}_p(R', T') \mid I' \not\models Q_{r,s,t}\}| = (2^r - 1)^{|R|-i} \times (2^t - 1)^{|T|-j} \times \alpha(i, j, c, d, d')^p$$

Where $\alpha(i, j, c, d, d')$ is defined as the following quantity:

$$\gamma^c \times \delta_{\mathsf{R}}^d \times \delta_{\mathsf{T}}^{d'} \times \delta_{\perp}^e \times \lambda_{\mathsf{R}}^{M_1(c+d)+M_2 i} \times \lambda_{\mathsf{T}}^{M_3 j} \times \bar{\lambda}_{\mathsf{R}}^{M_1(d'+e)+M_2(|R|-i)} \times \bar{\lambda}_{\mathsf{T}}^{M_3(|T|-j)}.$$

Let us substitute this value in Equation (5). Note that this value only depends on the cardinalities of R' and T' and the values of $\mathrm{c}, \mathrm{d}, \mathrm{d}', \mathrm{e}$, but not on the specific choice of R' and T'. Thus, splitting the sum accordingly, we can obtain the following:

▷ Claim 5.2. For any $0 \leq p < M$, we have that:

$$N_p = \sum_{0 \leq i \leq |R|} \sum_{0 \leq j \leq |T|} \sum_{0 \leq c,d,d' \leq m} Y_{i,j,c,d,d'} \times \alpha(i, j, c, d, d')^p$$

This equation can be expressed as a matrix equation $\vec{N} = A\vec{Y}$, with A the matrix whose cells contain $\alpha(i, j, c, d, d')^p$. Note that A is indeed an M-by-M matrix, where each row corresponds to a value of p with $0 \leq p < M$, and every column corresponds to a tuple (i, j, c, d, d'), for which there are M choices by definition of M. The matrix A relates the vector $\vec{N}$ computed from our oracle calls and the variables $\vec{Y}$ that we wish to determine to solve our problem on the graph G. It only remains to show that A is an invertible matrix, so that we can compute its inverse A^{-1} in polynomial time, use it to recover $\vec{Y}$ from $\vec{N}$, and from there recover P via Equation 4, concluding the reduction. Now, A is clearly a Vandermonde matrix, so we need just argue that its coefficients $\alpha(i, j, c, d, d')$ are different. We do this in the next section.

6 Showing that the Matrix is Invertible

In this section, we conclude our proof of Theorem 3.4 by showing the following:

▷ Claim 6.1. For all $(i, j, c, d, d') \neq (i_2, j_2, c_2, d_2, d'_2)$ with $0 \leq i, i_2 \leq |R|$, $0 \leq j, j_2 \leq |T|$, $0 \leq c, c_2, d, d_2, d', d'_2 \leq m$, we have $\alpha(i, j, c, d, d') \neq \alpha(i_2, j_2, c_2, d_2, d'_2)$.

This implies that the Vandermonde matrix A is invertible, and concludes the definition of the reduction and the proof of Theorem 3.4.

A Closer Look at γ, δ_{R}, δ_{T} and $\delta_{\perp}$. To show our claim, we will need to look deeper in the definition of α, which involves γ, δ_{R}, δ_{T} and $\delta_{\perp}$. Remember that these are the number of possible worlds of the $(*, *, *, *)$-gadgets defined in Section 4. To show the invertibility of the matrix, we will first need to understand what is the exponent of the number 2 in the decomposition of these numbers as a product of primes. This is abstracted away in the following lemma, which we prove by computing explicitly the numbers of possible worlds.

▶ **Lemma 6.2.** *The number γ is odd, and we have, for some odd quantities δ'_{R}, δ'_{T}, $\delta'_{\perp}$:*

$$\delta_{\mathsf{R}} = 2^s \times \delta'_{\mathsf{R}}$$
$$\delta_{\mathsf{T}} = 2^s \times \delta'_{\mathsf{T}}$$
$$\delta_{\perp} = (2^s)^2 \times \delta'_{\perp}$$

Second, we will need the following lemma on these quantities:

▶ **Lemma 6.3.** *For all $r, s, t \geq 1$, we have: $\delta_{\mathsf{R}} \times \delta_{\mathsf{T}} \neq \gamma \times \delta_{\perp}$.*

Proof sketch. We do a case distinction on the possible worlds accounted for in $\delta_{\mathsf{R}} \times \delta_{\mathsf{T}}$ and those accounted for in $\gamma \times \delta_{\perp}$, picking an order on the nodes that simplifies the comparison between the two case distinctions. By focusing on the cases where the number of possible worlds is different, we can explicitly compute the difference between these two quantities and show that it is non-zero, specifically it is $(2^s)^3 \times (2^r - 1) \times (2^t - 1)$. (We suspect that there may be a more elegant proof avoiding the need for this case distinction.) ◀

We can now use the previous lemmas to show Claim 6.1. Fix i, j, c, d, d' and i_2, j_2, c_2, d_2, d'_2. As usual, we denote $e = m - c - d - d'$ and $e_2 = m - c_2 - d_2 - d'_2$. By contraposition, we show that if $\alpha(i, j, c, d, d') = \alpha(i_2, j_2, c_2, d_2, d'_2)$ then the parameters are equal.

Equality on i and j, and Two Equations for c, d, d'. Let us rewrite the definition of α (given in Claim 5.1), using Lemma 6.2 and substituting the definition of $\bar{\lambda}_{\mathsf{R}}$ and $\bar{\lambda}_{\mathsf{T}}$ from Section 4:

$$\begin{aligned}\alpha(i,j,c,d,d') = \gamma^c \times (2^s)^d (\delta'_{\mathsf{R}})^d \times (2^s)^{d'} (\delta'_{\mathsf{T}})^{d'} \times (2^{2s})^e {\delta'_{\perp}}^e \\ \times \lambda_{\mathsf{R}}^{M_1(c+d)+M_2 i} \times \lambda_{\mathsf{T}}^{M_3 j} \times (2^{t+s})^{M_1(d'+e)+M_2(|R|-i)} \times (2^{r+s})^{M_3(|T|-j)}\end{aligned}$$

Recall that, by Lemma 6.2, the quantities γ, δ'_{R}, δ'_{T} and $\delta'_{\perp}$ are odd, and the quantities λ_{R} and λ_{T} as defined in Section 4 are odd. We get a similar equation for $\alpha(i_2, j_2, c_2, d_2, d'_2)$. From the integer equality $\alpha(i, j, c, d, d') = \alpha(i_2, j_2, c_2, d_2, d'_2)$, the coefficients of two in the prime number decompositions of these numbers must also be equal. This yields:

$$\begin{aligned}&s(d\ + d'\ + 2e\ ) + (t+s) \times (M_1(d'\ + e\ )) + (t+s) \times M_2(|R| - i\ ) + (r+s) \times M_3(|T| - j\ ) \\ &= s(d_2 + d'_2 + 2e_2) + (t+s) \times (M_1(d'_2 + e_2)) + (t+s) \times M_2(|R| - i_2) + (r+s) \times M_3(|T| - j_2)\end{aligned}$$

We now use the fact that, as $d, d', e \leq m$, we have $s(d + d' + 2e) \leq 4ms$. By definition of M_1, we have $s(d + d' + 2e) < M_1$. We also have $d' + e \leq 2m$, so that (using the previous inequality) we have $s(d + d' + 2e) + (t + s) \times (M_1(d' + e)) < M_2$ by definition of M_2. Last, we have $|R| - i \leq |R|$, so that (using the two previous inequalities) we have $s(d + d' + 2e) + (t + s) \times (M_1(d' + e)) + (t + s) \times M_2 \times (|R| - i) < M_3$ by the definition of M_3. Similar inequalities hold for the right-hand-side of the above equation. Thus, we can reason about the quotient of the equation by M_3, about the quotient by M_2 of its remainder modulo M_3, about the quotient by M_1 of the remainder modulo M_2 of the remainder modulo M_3, and about the remainder modulo M_1 of the remainder modulo M_2 of the remainder modulo M_3. This gives us four equations (where we also simplify by the constant factors s, $t + s$, $r + s$):

$$\begin{aligned}d + d' + 2e &= d_2 + d'_2 + 2e_2 \\ d' + e &= d'_2 + e_2 \\ |R| - i &= |R| - i_2 \\ |T| - j &= |T| - j_2\end{aligned}$$

The last two equations imply that $i = i_2$ and $j = j_2$, so we have shown that two quantities are equal, out of the five that define α. The two first equations imply $d' + e = d'_2 + e_2$ (second

equation), and $d + e = d_2 + e_2$ (subtracting the second equation from the first equation). Rewriting $e = m - c - d - d'$, rewriting e_2 likewise, and simplifying, we get:

$$c + d = c_2 + d_2$$
$$c + d' = c_2 + d'_2$$

These two equations do not suffice to justify that $(c, d, d') = (c_2, d_2, d'_2)$, so more reasoning is needed to get one additional equation and argue that these quantities must be equal.

Getting the Last Equation. Let us write the equality $\alpha(i, j, c, d, d') = \alpha(i_2, j_2, c_2, d_2, d'_2)$ and simplify all the (non-zero) values now known to be equal thanks to $i = i_2$, $j = j_2$:

$$\gamma^c \cdot (\delta_{\mathsf{R}})^d \cdot (\delta_{\mathsf{T}})^{d'} \cdot (\delta_\perp)^e \cdot \lambda_{\mathsf{R}}^{M_1(c+d)} \cdot \bar{\lambda}_{\mathsf{R}}^{M_1(d'+e)} = \gamma^{c_2} \cdot (\delta_{\mathsf{R}})^{d_2} \cdot (\delta_{\mathsf{T}})^{d'_2} \cdot (\delta_\perp)^{e_2} \cdot \lambda_{\mathsf{R}}^{M_1(c_2+d_2)} \cdot \bar{\lambda}_{\mathsf{R}}^{M_1(d'_2+e_2)}$$

The previously shown equations also imply that the λ_{R} and $\bar{\lambda}_{\mathsf{R}}$ factors simplify, so we get:

$$\gamma^{c-c_2} \times (\delta_{\mathsf{R}})^{d-d_2} \times (\delta_{\mathsf{T}})^{d'-d'_2} \times (\delta_\perp)^{e-e_2} = 1$$

The equation $c + d = c_2 + d_2$ (shown above) implies that $c - c_2 = d_2 - d$, and subtracting that equation from $c + d' = c_2 + d'_2$ (shown above) gives $d' - d = d'_2 - d_2$, so that $d' - d'_2 = d - d_2$. As we know $d + e = d_2 + e_2$ (above), we have $e - e_2 = d_2 - d$. So using $c - c_2 = d_2 - d$, $d' - d'_2 = d - d_2$, and $e - e_2 = d_2 - d$, we have:

$$(\gamma \times \delta_\perp)^{d_2-d} = (\delta_{\mathsf{R}} \times \delta_{\mathsf{T}})^{d_2-d}$$

Now, by Lemma 6.3, we have $\gamma \times \delta_\perp \neq \delta_{\mathsf{R}} \times \delta_{\mathsf{T}}$. Thus, this equation implies that $d = d_2$. In combination with the equations that we showed above, this completely specifies the system: from $c + d = c_2 + d_2$ we get $c = c_2$, and from $c + d' = c_2 + d'_2$ we get $d' = d'_2$. Thus, we have $(c, d, d', i, j) = (c_2, d_2, d'_2, i_2, j_2)$. This establishes Claim 6.1 and shows that all coefficients $\alpha(c, d, d', i, j)$ of the Vandermonde matrix A are different, so it is invertible. This concludes the proof of Theorem 3.4, and hence of our main result (Theorems 3.1 and 3.2).

7 Extending to Uniform Probabilities

Having proved our main result (Theorem 3.1), we now turn to a natural variant of the probabilistic query evaluation problem: what if, instead of imposing that all probabilities are $\frac{1}{2}$ (which amounts to uniform reliability), we impose that all tuples of the same relation have the same probability? Let us formally define this variant:

▶ **Definition 7.1.** *Let Q be a CQ without self-joins, and let φ be a function mapping each relation symbol ρ of Q to a rational number $0 < \varphi(\rho) \leq 1$. The problem $\mathsf{PQE}_\varphi(Q)$ is the problem $\mathsf{PQE}(Q)$ on input instances I over Q whose probability function $\pi : I \to [0, 1]$ is defined by φ, i.e., for every fact $f \in I$, we have $\pi(f) = \varphi(\rho)$ where ρ is the relation symbol used in f.*

In this section, we will focus on this problem for the hard query $Q_1 : \mathsf{R}(x), \mathsf{S}(x, y), \mathsf{T}(y)$ from Equation 1, and write the problem directly as $\mathsf{PQE}_{r,s,t}(Q_1)$ where $0 < r, s, t \leq 1$ are the respective images of R, S and T under φ. In this language, the uniform reliability problem for Q_1 is (up to renormalization) the problem $\mathsf{PQE}_{\frac{1}{2},\frac{1}{2},\frac{1}{2}}(Q_1)$, which we have shown to be #P-hard in Theorem 3.1 because Q_1 is non-hierarchical. The usual #P-hardness proof for

$\mathsf{PQE}(Q_1)$ [6], where we reduce from the problem of counting the satisfying assignment of a monotone partitioned DNF formula [19], is actually a hardness proof for $\mathsf{PQE}_{\frac{1}{2},1,\frac{1}{2}}(Q_1)$. The natural question is whether hardness can be shown for other values of r, s, and t.

Let us first observe that we can use our main hardness result on uniform reliability (Theorem 3.2) to show hardness in a specific case via an easy reduction. While we do not hope that the technique can generalize, it still classifies some of the cases:

▶ **Corollary 7.2.** $\mathsf{PQE}_{2^{-r},2^{-s},2^{-t}}(Q_1)$ *is #P-hard for all natural numbers* $r, s, t > 0$.

Proof. Consider the query $Q_{i,j,k}$, following Equation 2. We have shown in Theorem 3.4 that uniform reliability for $Q_{i,j,k}$ is #P-hard. We reduce this problem to the PQE variant that we consider. To do this, consider an input instance I to $\mathsf{UR}(Q_{i,j,k})$. We call an element a of I *useless for* R_* if some R_*-fact does not hold on a, we define a being *useless for* T_* analogously, and we call a pair (a, b) of I *useless for* S_* if some S_*-fact does not hold about the pair. We call an R_*-fact *useless* if it holds about an element that is useless for R_*, and extend this definition to T_*-facts and to S_*-facts (for element pairs). It is clear that no match of $Q_{i,j,k}$ on I can involve a useless fact. Indeed, when a match of the query involves a fact f of the form $\mathsf{R}_*(a)$, then the match witnesses that all other R_*-facts hold on a, so that a is not useless for R_*, and f is not useless. The same reasoning applies to S_*-facts and T_*-facts.

Hence, let us compute in linear time the subinstance I' of I where we only keep the facts that are not useless: this is doable in polynomial time. Now, any subset of I is defined by picking a subset J' of I' and a subset J'' of $I \setminus I'$. Further, as $I \setminus I'$ only consists of useless facts, it is clear that if some subset J' does not satisfy $Q_{i,j,k}$ then $J := J' \cup (I \setminus I')$ still does not, because the facts added in $I \setminus I'$ cannot be part of a query match in I, hence in J. All this reasoning shows that the answer to $\mathsf{UR}(Q_{i,j,k})$ on I is the answer to the same problem on I', multiplied by the number of possible choices for J'', that is, $2^{|I \setminus I'|}$. Hence, to show hardness, it suffices to reduce the uniform reliability problem on I' to our PQE problem.

Now, we can rewrite I' in linear time to I'' by replacing the set of R_*-facts on every element a where they exist by a single R-fact, and doing the same for T_*-facts and for S_*-facts (on element pairs). As we have removed useless facts, all facts of I' are thus taken into account in the rewriting. Now, define the probability assignment π on I'' by mapping the R, S, and T-facts to 2^{-i}, 2^{-j}, and 2^{-k} respectively. This means that (I'', π) is an instance to the $\mathsf{PQE}_{2^{-i},2^{-j},2^{-k}}(Q_1)$ problem that we are reducing to. Now, there is a clear correspondence from the subsets of I' to the possible worlds of I'', which is defined by rewriting to the signature R, S, T as we did in the reduction; and the number of preimages of each possible world of I'' is exactly equal to its probability according to π, up to renormalization by a constant factor of $2^{-|I'|}$. Thus, the answer to $\mathsf{UR}(Q_{i,j,k})$ on I' is exactly the answer to $\mathsf{PQE}_{2^{-i},2^{-j},2^{-k}}(Q_1)$ on I'' up to renormalization. This concludes the proof. ◀

Note that this does not classify the complexity of $\mathsf{PQE}_{r,s,t}(Q_1)$ for arbitrary values of r, s, t. Conversely, $\mathsf{PQE}_{r,s,1}(Q_1)$ and $\mathsf{PQE}_{1,s,t}(Q_1)$ are clearly solvable in polynomial time:

▶ **Proposition 7.3.** $\mathsf{PQE}_{r,s,1}(Q_1)$ *and* $\mathsf{PQE}_{1,s,t}(Q_1)$ *are in PTIME for all* $0 < r, s, t \leq 1$.

Proof sketch. When the R-facts have probability 1, we can rewrite Q_1 to $\mathsf{S}(x, y), \mathsf{T}(y)$ which is hierarchical, hence safe (Theorem 2.2). The other case is symmetric. ◀

We conjecture that these are the only tractable cases. Specifically, we conjecture the following generalization of the #P-hardness of $\mathsf{UR}(Q_1)$:

▶ **Conjecture 7.4.** $\mathsf{PQE}_{r,s,t}(Q_1)$ *is #P-hard for all* $0 < r, s, t \leq 1$ *with* $r < 1$ *and* $t < 1$.

We leave this for future work, but can prove Conjecture 7.4 in the case of $s = 1$. Formally:

▶ **Theorem 7.5.** $\mathsf{PQE}_{r,1,t}(Q_1)$ *is #P-hard for every* $0 < r < 1$ *and* $0 < t < 1$.

Proof sketch. Like in the previous section, we reduce from the problem of counting the independent sets of a bipartite graph. We do a simpler coding using simpler gadgets, and we show like before that the number of independent sets (parameterized by their number of vertices to the left and right) can be connected to the answer to the uniform reliability problem on a family of instances with different gadget instantiations. To argue that the matrix of the equation system is invertible, we notice that it is the Kronecker product of two invertible Vandermonde matrices. ◀

8 Conclusion

While query evaluation over TIDs has been studied for over a decade, the basic case of a uniform distribution, namely uniform reliability, had been left open. We have settled this open question for the class of CQs without self-joins, and shown a dichotomy on computational complexity of counting satisfying database subsets: this task is tractable for hierarchical queries, and #P-hard otherwise. We have also embarked on the investigation of the more general variant of CQ evaluation over TIDs with uniform probabilities. We have shown tractability for some combinations of probabilities by a straightforward reduction from uniform reliability, and shown hardness for others using different proof techniques.

In future work, we plan to investigate whether and how the proof of our main result can be simplified, with the goal of showing hardness for all combinations of uniform probabilities not known to be tractable, at least for the query Q_1 (Conjecture 7.4). An interesting special case is when these probabilities are $1/2$ and 1, that is, some relations are deterministic while the others define a uniform distribution over all their subsets; note that hardness in this setting is a stronger statement than hardness in the model of Dalvi and Suciu [7], as in this model the tuples of the probabilistic relations can have arbitrary probabilities.

Another question is whether our results could extend to more general query classes. A natural question would be to study the question for CQs with self-joins. More generally, we could study the case of UCQs with self-joins, and try to match the known dichotomy for non-uniform probabilities [7]. Following our work, considerable progress in this direction has been done recently by Kenig and Suciu [14], which addresses the case of PQE with probabilities of $1/2$ and 1, and leaves open the case of uniform reliability for arbitrary UCQs.

References

1 Antoine Amarilli and Benny Kimelfeld. Uniform reliability of self-join-free conjunctive queries. In *ICDT*, 2021.

2 Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. Symmetric weighted first-order model counting. In *PODS*, pages 313–328. ACM, 2015.

3 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *PODS*, pages 303–318. ACM, 2017.

4 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34:1–34:41, 2013.

5 Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. From weighted to unweighted model counting. In *IJCAI*, 2015.

6 Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4):523–544, 2007.

7 Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2012.

8 Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: Diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

9 Pradeep Dubey and Lloyd S. Shapley. Mathematical properties of the Banzhaf power index. *Mathematics of Operations Research*, 4(2):99–131, 1979.

10 Erich Grädel, Yuri Gurevich, and Colin Hirsch. The complexity of query reliability. In *PODS*, pages 227–234. ACM Press, 1998.

11 Eric Gribkoff and Dan Suciu. SlimShot: In-database probabilistic inference for knowledge bases. *PVLDB*, 9(7):552–563, 2016.

12 B. Grofman and H. Scarrow. *Iannucci and Its Aftermath: The Application of the Banzhaf Index to Weighted Voting in the State of New York*, pages 168–183. Physica-Verlag HD, Heidelberg, 1979. doi:10.1007/978-3-662-41501-6_10.

13 Abhay Kumar Jha and Dan Suciu. Probabilistic databases with MarkoViews. *PVLDB*, 5(11):1160–1171, 2012.

14 Batya Kenig and Dan Suciu. A dichotomy for the generalized model counting problem for unions of conjunctive queries. *CoRR*, abs/2008.00896, 2020.

15 Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. The Shapley value of tuples in query answering. In *ICDT*, volume 155 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

16 Dany Maslowski and Jef Wijsen. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.*, 79(6):958–983, 2013.

17 Dany Maslowski and Jef Wijsen. Counting database repairs that satisfy conjunctive queries with self-joins. In *ICDT*, pages 155–164. OpenProceedings.org, 2014.

18 Dan Olteanu and Jiewen Huang. Using OBDDs for efficient query evaluation on probabilistic databases. In *SUM*, volume 5291 of *Lecture Notes in Computer Science*, pages 326–340. Springer, 2008.

19 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4), 1983.

20 Alvin E Roth. *The Shapley value: Essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

21 Babak Salimi, Leopoldo E. Bertossi, Dan Suciu, and Guy Van den Broeck. Quantifying causal effects on query answering in databases. In *TAPP*, 2016.

22 L.S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39:1095–1100, 1953.

23 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

Efficient Differentially Private F_0 Linear Sketching

Rasmus Pagh
University of Copenhagen, Denmark
BARC, Copenhagen, Denmark

Nina Mesing Stausholm
IT University of Copenhagen, Denmark
BARC, Copenhagen, Denmark

Abstract

A powerful feature of *linear sketches* is that from sketches of two data vectors, one can compute the sketch of the difference between the vectors. This allows us to answer fine-grained questions about the difference between two data sets. In this work we consider how to construct sketches for weighted F_0, i.e., the summed weights of the elements in the data set, that are small, differentially private, and computationally efficient. Let a weight vector $w \in (0,1]^u$ be given. For $x \in \{0,1\}^u$ we are interested in estimating $\|x \circ w\|_1$ where $\circ$ is the Hadamard product (entrywise product).

Building on a technique of Kushilevitz et al. (STOC 1998), we introduce a sketch (depending on w) that is linear over GF(2), mapping a vector $x \in \{0,1\}^u$ to $Hx \in \{0,1\}^\tau$ for a matrix H sampled from a suitable distribution $\mathcal{H}$. Differential privacy is achieved by using *randomized response*, flipping each bit of Hx with probability $p < 1/2$. That is, for a vector $\varphi \in \{0,1\}^\tau$ where $\Pr[(\varphi)_j = 1] = p$ independently for each entry j, we consider the *noisy sketch* $Hx + \varphi$, where the addition of noise happens over GF(2). We show that for every choice of $0 < \beta < 1$ and $\varepsilon = O(1)$ there exists $p < 1/2$ and a distribution $\mathcal{H}$ of linear sketches of size $\tau = O(\log^2(u)\varepsilon^{-2}\beta^{-2})$ such that:

1. For random $H \sim \mathcal{H}$ and noise vector φ, given $Hx + \varphi$ we can compute an estimate of $\|x \circ w\|_1$ that is accurate within a factor $1 \pm \beta$, plus additive error $O(\log(u)\varepsilon^{-2}\beta^{-2})$, w. p. $1 - u^{-1}$, and
2. For every $H \sim \mathcal{H}$, $Hx + \varphi$ is ε-differentially private over the randomness in φ.

The special case $w = (1, \ldots, 1)$ is *unweighted* F_0. Previously, Mir et al. (PODS 2011) and Kenthapadi et al. (J. Priv. Confidentiality 2013) had described a differentially private way of sketching unweighted F_0, but the algorithms for calibrating noise to their sketches are not computationally efficient, either using quasipolynomial time in the sketch size or superlinear time in the universe size u.

For fixed ε the size of our sketch is polynomially related to the lower bound of $\Omega\left(\log(u)\beta^{-2}\right)$ bits by Jayram & Woodruff (Trans. Algorithms 2013). The additive error is comparable to the bound of $\Omega\left(1/\varepsilon\right)$ of Hardt & Talwar (STOC 2010). An application of our sketch is that two sketches can be added to form a noisy sketch of the form $H(x_1 + x_2) + (\varphi_1 + \varphi_2)$, which allows us to estimate $||(x_1 + x_2) \circ w||_1$. Since addition is over GF(2), this is the weight of the symmetric difference of the vectors x_1 and x_2. Recent work has shown how to privately and efficiently compute an estimate for the symmetric difference size of two sets using (non-linear) sketches such as FM-sketches and Bloom Filters, but these methods have an error bound no better than $O(\sqrt{\bar{m}})$, where $\bar{m}$ is an upper bound on $||x_1||_0$ and $||x_2||_0$. This improves previous work when $\beta = o\left(1/\sqrt{\bar{m}}\right)$ and $\log(u)/\varepsilon = \bar{m}^{o(1)}$.

In conclusion our results both improve the efficiency of existing methods for unweighted F_0 estimation and extend to a weighted generalization. We also give a distributed streaming implementation for estimating the size of the union between two input streams.

2012 ACM Subject Classification Security and privacy → Formal methods and theory of security

Keywords and phrases Differential Privacy, Linear Sketches, Weighted F0 Estimation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.18

Related Version *Full Version*: https://arxiv.org/pdf/2001.11932.pdf

Funding This work was supported by Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

Acknowledgements We thank Shuang Song and Abhradeep Guha Thakurta for feedback on a previous version of this manuscript. We thank the anonymous reviewers for constructive suggestions. The work of Rasmus Pagh was partly done while employed at IT University of Copenhagen.

1 Introduction

Estimating the number of distinct values in a set (its *cardinality*), without explicitly enumerating the set, is a classical and important problem in data management. Sampling-based methods [22] can in many cases be improved by using algorithms designed with data streams in mind [25]. Streaming algorithms based on *linear sketches* can also be used to estimate changes as a data set evolves [27] and for approximate query processing in distributed settings [3, 13]. As our first motivating example consider the following SQL query:

```
SELECT P.name
FROM EMPLOYEES E, HOSPITALIZATION H
WHERE E.salary > 100000 AND E.name = H.name AND H.year = 2020
```

The size (in bytes) of the query result is a sum weighted by string length over the names that appear in subsets of two relations. That is, estimating the size of the join result is about estimating the *weighted* size of a set intersection.

In recent years, *privacy* of database records has become increasingly important when releasing aggregates from a database. In the example above, the information that a tuple with a particular person exists (and satisfies a certain predicate) can potentially be sensitive. If the database is distributed, with relations on different servers that are not allowed to expose sensitive information, it is not trivial how to even estimate the join size.

The notion of *differential privacy* [15] has emerged as the leading approach to providing rigorous privacy guarantees. It is known that differential privacy comes with pitfalls [28], but work in the database community has led to privacy-preserving database systems supporting (limited) SQL, see e.g. [32, 46] and their references. A challenge in such systems is that the set of queries is often not known ahead of time, so *budgeting* the disclosure of detailed information is highly nontrivial. An attractive approach to achieving privacy even when faced with unknown queries is to release a summary, or sketch, of the data set from which approximate answers to queries can be computed (as a side effect this also eliminates the need for interaction). In this paper we consider private linear sketches for the problem of cardinality estimation.

Example. Suppose that the company Acme Corporation runs an employee satisfaction survey once a year. Management at Acme Corporation made some drastic changes over the past year, and they wish to analyze the impact of these changes on the employees' satisfaction. For a specific improvement, every employee is given a value between 0 and 1, indicating how closely related that improvement is to the employee's work life. A survey for each improvement is run by a consultant who delivers a summary of the results to the management at Acme Corporation. The consultant ensures that the summary is private, so individual employees cannot be identified from the summary. The management at Acme Corporation can combine the summary from last year's survey with the summary from this year's survey to estimate the change in satisfaction over the past year, where the vote of an employee is weighted by the value that employee was given. We note that the summaries should be generated in the same way, but the choice of consultant may change from year to year.

More formally, we consider two players that hold sets A and B from a universe $U = \{1, ..., u\}$, respectively. For every element $j \in U$ let a fixed, public weight, $w_j \in (0, 1]$ be given and for input set $A \subseteq U$ consider the corresponding weight vector $(w_A)_j = w_j \cdot \mathbf{1}[j \in A]$. The goal is to estimate the weight of the symmetric difference $\|w_{A \triangle B}\|_1$, in a differentially private manner. We refer the reader to Section 3.2 for the basics of differential privacy. One may think of the sets as two lists of employees. Given input sets A and B, the two players each compute a *linear sketch* of their own set and add noise to obtain privacy as described in Section 4. These noisy sketches can be thought of as the summaries.

For input sets A and B, we note that if we, along with the estimate of the weight of the symmetric difference, have estimates of $\|w_A\|_1$ and $\|w_B\|_1$, then we can also estimate $\|w_{A \cup B}\|_1$, $\|w_{A \cap B}\|_1$, $\|w_{A \setminus B}\|_1$ and $\|w_{B \setminus A}\|_1$ as argued in Section 4.3. To make this possible, each player also outputs a differentially private version of their set weight. We remark that if all weights $w_j = 1$, then the problem reduces to estimating the set *size*, a problem often referred to as F_0.

We remark that it is not clear how to estimate $\|w_{A \triangle B}\|_1$ from $\|w_A\|_1$ and $\|w_B\|_1$, and so it seems insufficient to have each party simply compute and release private versions of $\|w_A\|_1$ and $\|w_B\|_1$. We define and construct a noisy linear sketch over GF(2), the field of size 2, with the following properties:

- ε-differentially private
- Computationally efficient
- Allows estimating the weight of the symmetric difference with small relative error
- Space usage is polynomially related to the lower bound (for fixed ε)

Previously known results satisfy at most 3 of these properties, see Table 1 for an overview. We discuss previous work further in Section 2. Our sketch can be computed and stored for future use, meaning that two players do not have to be active simultaneously but can compute and publish their sketches when they are ready. A self-contained description of our linear sketch can be found in Section 4. Readers familiar with the sketching literature will realize that our sketch combines a method of Kushilevitz, Ostrovsky, and Rabani [29] with a standard hashing-based subsampling technique (see, e.g., [47]), and we use a Randomized Response Technique [45] with noise parameter $p(\varepsilon)$, to get ε-differential privacy. Hence, refer to our sketch as the *KOR sketch* and to its noisy counterpart as a *noisy KOR sketch*. We note that a related, but non-linear and non-private, sketch has previously been used for estimating size of symmetric difference [38]. From now on we leave out ε in the noise parameter and write simply p. We show that the KOR sketch is sufficiently robust to noise to allow precise estimation after adding noise, thus allowing pure differential privacy.

We next give an overview of our techniques, discussed in depth in Section 4. Let $U = \{1, ..., u\}$ be the universe from which the input sets are taken. Privacy parameter ε and accuracy parameter β are given, and a sketch size τ is determined by these parameters. We show in Section 5.2 that we can construct an ε-differentially private sketch from which we can compute a $(1 + \beta)$-approximation for the weight of the symmetric difference with high probability.

Randomized response [45] is applied to the entire sketch Hx, meaning that each entry of the sketch is flipped with probability $p < 1/2$. We show in Section 5.1 how to choose p as a function of ε to ensure ε-differential privacy for the sketch. Let $x \circ w$ denote the Hadamard product. Our main theorem is:

▶ **Theorem 1** (Noisy KOR sketch). *Let $w \in (0, 1]^u$ be given. For every choice of $0 < \beta < 1$ and $\varepsilon = O(1)$ there exists a distribution $\mathcal{H}$ over GF(2)-linear sketches mapping a vector $x \in \{0, 1\}^u$ to $\{0, 1\}^\tau$, where $\tau = O\left(\log^2(u)\varepsilon^{-2}\beta^{-2}\right)$, and a distribution $\mathcal{N}_\varepsilon$ over noise vectors such that:*

Table 1 Selected lower bounds (top part) and upper bounds (bottom part) for estimating the (unweighted) size of the symmetric difference $m = |A \triangle B|$ from small sketches of sets $A, B \subseteq \{1, \ldots, u\}$. Bounds stated as $\tilde{O}$ and $\tilde{\Omega}$ are simplified by suppressing multiplicative factors polynomial in $\log(1/\varepsilon)$, $\log(1/\beta)$, $\log(1/\delta)$, and $\log u$. The non-private bounds in [24, 25] improve previous results by an $\tilde{O}(1)$ factor, we refer to their references for details. * The space usage of [26] is measured in terms of real numbers; it is unclear how much space a private, discrete implementation would need.

Reference	DP	Additive error	Rel. error	Initial. time	Space usage
Hardt & Talwar [23]	ε	$\Omega(1/\varepsilon)$	–	–	–
McGregor et al. [30]	ε	$\tilde{\Omega}(\sqrt{m}/e^{\varepsilon})$	–	–	–
Jayram & Woodruff [24]	–	–	$1+\beta$	–	$\tilde{\Omega}(1/\beta^2)$
Kane et al. [25]	–	$\tilde{O}(1)$	$1+\beta$	$O(1)$	$\tilde{O}(1/\beta^2)$
Mir et al. [35]	ε	$\tilde{O}(m^{1-\Omega(1)}/\varepsilon^{O(1)})$	$1+\beta$	$\exp((\varepsilon\beta)^{-O(1)})$	$\tilde{O}((\varepsilon\beta)^{-O(1)})$
Kenthapadi et al. [26]	(ε, δ)	$\tilde{O}(\sqrt{m}/\varepsilon)$	$1+\beta$	$\tilde{\Omega}(u)$	$\tilde{O}(1/\beta^2)^*$
Stanojevic et al. [41]	ε	$\tilde{O}(\sqrt{\lvert A \cup B\rvert}/\varepsilon^2)$	–	$\Omega(\lvert A\rvert + \lvert B\rvert)$	$\Omega(\lvert A\rvert + \lvert B\rvert)$
This paper	ε	$\tilde{O}(m^{2/3}/\varepsilon^{2/3})$	$1+\beta$	$\tilde{O}(\varepsilon^{-2}\beta^{-2})$	$\tilde{O}\left(\varepsilon^{-2}\beta^{-2}\right)$

1. *For $H \sim \mathcal{H}$ and $\varphi \sim \mathcal{N}_\varepsilon$, given $Hx + \varphi$ we can compute, in time $O(\tau)$, an estimate $\hat{w}$ of $\|x \circ w\|_1$ that with probability $1 - 1/u$ satisfies $|\hat{w} - \|x \circ w\|_1| < \beta\|x \circ w\|_1 + O\left(\log(u)\varepsilon^{-2}\beta^{-2}\right)$.*
2. *For every H in the support of $\mathcal{H}$, $Hx + \varphi$ is ε-differentially private over the choice of $\varphi \sim \mathcal{N}_\varepsilon$, and can be computed in time $O(\|x\|_0 \log(u) + \tau)$, including time for sampling φ.*

The assumption that $\varepsilon = O(1)$ is not essential, and is only made to simplify our bounds (which do not improve for privacy parameter $\varepsilon = \omega(1)$). Without loss of generality we can assume that parameter β is such that the error is dominated by $\beta\|x \circ w\|_1$, because reducing β further cannot reduce error by more than a factor 2. In the unweighted case, setting $\beta = \sqrt[3]{\log(u)/(\varepsilon^2 m)}$ to balance relative and additive error we get error $\tilde{O}(m^{2/3}/\varepsilon^{2/3})$, where the $\tilde{O}$ notations suppresses a polylogarithmic factor. This is polynomially related to known lower bounds described in section 2.3.

Applications

Suppose that Alice holds set A with corresponding characteristic vector $x_A \in \{0,1\}^u$ and Bob holds set B with characteristic vector $x_B \in \{0,1\}^u$. They jointly sample $H \sim \mathcal{H}$ and privately sample $\varphi_A, \varphi_B \sim \mathcal{N}_\varepsilon$ according to Theorem 1. Then $Hx_A + \varphi_A$ and $Hx_B + \varphi_B$ are ε-differentially private. Furthermore, $(Hx_A + \varphi_B) + (Hx_B + \varphi_B) = (Hx_A + Hx_B) + (\varphi_A + \varphi_B)$, and we show in Section 4.3 that $\varphi_A + \varphi_B \sim \mathcal{N}_{\varepsilon'}$ with $\varepsilon' = \varepsilon^2/(2 + 2\varepsilon)$. In Section 5.2 we use this in conjunction with Theorem 1 to establish:

▶ **Corollary 2.** *For accuracy parameter $\beta > 0$, consider an ε-differentially private noisy KOR sketch for a set A and an ε-differentially private noisy KOR sketch for a set B, based on the same linear sketch $H \sim \mathcal{H}$, sampled independently of A and B. We can compute an approximation $\hat{\Delta}$ of the weight of the symmetric difference, such that with probability $1 - 1/u$:*

$$|\|w_{A \triangle B}\|_1 - \hat{\Delta}| < \beta\|w_{A \triangle B}\|_1 + \mathrm{poly}(1/\varepsilon, 1/\beta, \log u) \ .$$

In the special case where all weights w_j are 1, this reduces to estimating the *size* of the symmetric difference $A \triangle B$.

In Section 6 we describe how to modify our sketch to apply in a streaming setting. In this case, we estimate the size of the union of the input streams rather than the size of the symmetric difference when merging two sketches.

2 Related Work

In the absence of privacy constraints, seminal estimators for (unweighted) set cardinality that support merging sketches (to produce a sketch of the union) are HyperLogLog [19], FM-sketches [20], and bottom-k (aka. k-minimum values) sketches [6]. Progress on making these estimators private for set operations include [42] (using FM-sketches) and [40], which builds a private cardinality estimator to estimate set intersection size using the bottom-k sketch. We note that these sketches do not achieve differential privacy, but are aimed at a weaker notion of privacy. Specifically, they offer a one-sided guarantee that may reveal that an individual element is *not* present in the dataset. To our best knowledge, a private version of HyperLogLog with provable bounds on accuracy has not been described in the literature.

The *weighted* version of cardinality estimation has been less studied. For (scaled) integer weights in $[W]$ there is a simple reduction that inserts element i with weight w_i by inserting the tuples $(i,1),\ldots,(i,w_i)$ into a standard cardinality estimator on the domain $U \times [W]$, but this makes the obtained bounds depend on the number W of possible weights. Cohen et al. [12] showed that the class of cardinality estimators that rely on extreme order statistics (for example HyperLogLog) can be efficiently extended to the weighted setting, even for real-numbered weights.

Note that the weighted F_0 estimation problem is different from F_1 and L_1 estimation in the context of set operations, for example, the union of two identical sets will have the same weighted F_0, whereas summing two identical vectors will produce a vector with twice the L_1 norm. In the rest of this section we focus on the standard, unweighted setting.

2.1 Differentially private cardinality estimators

Already the seminal paper on pan-privacy [16] discusses differentially private streaming algorithms for F_0 on insertion-only streams. Their sketch is not linear and does not allow deletions or subtraction of sketches. It is not clear if the sketch can be merged to produce a sketch for the union. Recent work by von Voigt et al. [44] has shown how to estimate the cardinality of a set using less space in a differentially private manner using FM-sketches, using the Probabilistic Counting with Stochastic Averaging (PCSA) technique [20]. These sketches can be merged to obtain a sketch for the union of the input set with a slightly higher level of noise. Privacy is achieved by randomly adding ones to the sketch and by only sketching a sample of the input dataset.

Bloom Filters have been studied extensively to obtain cardinality estimators under set operations (already implicit in [16]). Alaggan et al. [2] estimated set intersection size by combining a technique for computing similarity between sets, represented by Bloom filters in a differentially private manner, named BLIP (BLoom-then-flIP) filters [1] with a technique for approximating set intersection of two sets based on their Bloom Filter representation [9]. We note that [1] achieves privacy by flipping each bit of the Bloom filter with a certain probability, much like the technique we use to get privacy of our sketch. Stanojevic et al. [41] show how to estimate set intersection, union and symmetric difference for two sets by computing an estimate for the size of the union, and combined with the size of each set, they show how to compute an estimate for the size of the intersection and the symmetric

difference. They achieve privacy by flipping each bit with some probability, like in [1]. Also, RAPPOR [18] uses Bloom Filters with a Randomized Response technique to collect data from users in a differentially private way but is mainly aimed at computing heavy hitters.

Though a bound on the expected worst-case error of privately estimating the size of a symmetric difference $|A \triangle B|$ (as in Corollary 2) is not stated in any of these papers, an upper bound of $O(\sqrt{\bar{m}})$, where $\bar{m}$ is an upper bound on the size of the sets, follows from the discussion in [41] (for fixed ε). It seems that this magnitude of error is inherent to approaches using Bloom filters since it arises by balancing the error related to the noise and the error related to hash collisions in the Bloom filter. An advantage and special case of our noisy KOR sketch is that it can be used to directly estimate the size of the symmetric difference, and so the error will depend only on the size of the symmetric difference. It seems that with non-linear sketches it would be necessary to first estimate the size of the union and combine this with the size of each input set as exhibited in, for example, [41]. Hence, the error would depend on the size of the union of the input sets.

2.2 Differentially private sketches

Closely related to our work is the differentially private Johnson-Lindenstrauss (JL) sketch by Kenthapadi et al. [26], in which the technique of adding noise to the sketch is also applied. Kenthapadi et al. add Gaussian noise, so to store and maintain a sketched vector, some kind of discretization would be needed (not discussed in their paper). Discretizing a real-valued private mechanism is non-trivial: Without sufficient care, one might lose privacy due to rounding in an implementation, as argued by Mironov [36]. Even if a suitable discretization of the mechanism in [26] would be possible (see [10] for a general discussion), it has several drawbacks compared to our method:

- It only achieves *approximate* differential privacy as opposed to the pure differential privacy of the noisy KOR sketch.
- The time needed to update the sketch when a set element is inserted or removed is not constant (in the main method described it is linear in the sketch size).
- The time needed to initialize the sketch is linear in the size of the sketch matrix, which has u columns, because the noise needs to be calibrated to the sensitivity of the JL sketch matrix, which requires linear time in the size of the sketch matrix. Alternatively, which is the suggestion in Kenthapadi et al., the sketch matrix is assumed to have low sensitivity and noise is calibrated to this sensitivity. If a sketch matrix with a large entry is randomly chosen, the sensitivity of the sketch matrix is large, in which case the noise does not ensure privacy. So with a small probability, privacy is not preserved.

Another closely related work is the paper of Mir et al. [35], which also adds a noise vector after computing standard linear sketches for F_0 estimation to make the sketch differentially private. They further initialize their sketches with random noise vectors to also get pan-privacy. The error bound obtained is similar to ours, and the sketch has a discrete representation, but their method is inferior in terms of time complexity. This is because they rely on the *exponential mechanism* [31], which is not computationally efficient. (Note that a preprint of the paper of Mir et al. [34] presented a computationally more efficient method. However, the sensitivity analysis in that paper has an error [39] that was corrected in the slower method published in [35].)

Our method is more computationally efficient and arguably simpler than the methods of [26, 35]. Our linear sketch is not a replacement for these sketches, though, since our sketch is over GF(2) rather than the reals (or integers).

2.3 Lower bounds

Jayram and Woodruff [24] show that, even with no privacy guarantee, to obtain error probability $1/u$ we need a sketch of $\Omega\left(\log(u)\beta^{-2}\right)$ bits to estimate F_0 with relative error $1 \pm \beta$. It is easy to extend this lower bound to our setting, in which an additive error of c is allowed: Simply insert each item c times, to increase the size of the set so that the additive error is negligible. Formally this requires us to extend the universe to $U \times \{1, \ldots, c\}$, such that the lower bound in terms of the original universe size becomes $\Omega\left(\log(u/c)\beta^{-2}\right)$. (The reason why we do not use this reduction to eliminate the additive error in our upper bound is that the reduction increases the sensitivity of updates, destroying the differential privacy properties.)

Hardt and Talwar [23] show that an ε-differentially private sketch for F_0 must have additive error $\Omega(1/\varepsilon)$, which is comparable (up to polynomial and logarithmic factors) to the additive error we achieve.

Desfontaines et al. [14] show that it is not possible to preserve privacy in accurate cardinality estimators if we can merge several sketches without loss in accuracy. Our sketch will have an increase in noise when merging sketches, and thus does not satisfy the requirement for cardinality estimators formulated in [14].

McGregor et al. [30] showed that in order to estimate the size of the intersection of two sets A and B, based on differentially private sketches of A and B, an additive error of $\Omega(\sqrt{u}/e^{\varepsilon})$ is needed in the worst case when A and B are arbitrary subsets of $[u]$. The lower bound holds even in an interactive setting where Alice (holding A) and Bob (holding B) can communicate, and we require that the communication transcript is differentially private. The hard input distribution uses sets with symmetric difference of size $\Theta(u)$ with high probability. Since $|A \cap B| = (|A| + |B| - |A \triangle B|)/2$, estimating the intersection size is no more difficult (up to constant factors in error) than estimating $|A|$, $|B|$, and $|A \triangle B|$. We can estimate $|A|$ and $|B|$ with error $O(1/\varepsilon)$ under differential privacy, so it follows that estimating $|A \triangle B|$ under differential privacy requires error $\Omega(\sqrt{u}/e^{\varepsilon})$. For a contrasting upper bound, [43, 37] suggest an algorithm estimating two-party set intersection size up to an additive error of $O(\sqrt{u}/\varepsilon)$ with high probability. A lower bound in terms of the size m of the symmetric difference follows by setting $u = m$.

2.4 Noisy sketching

In addition to the paper of Mir et al. [35], there is some previous work on sketching techniques in the presence of noise. Motivated by applications in learning theory, Awasthi et al. [5] considered recovery of a vector based on noisy 1-bit linear measurements. The resistance to noise demonstrated is analogous to what we show for the KOR sketch, but technically quite different since the linear mapping is computed over the reals before a sign operation is applied.

In a very recent paper [11], Choi et al. propose a framework for releasing differentially private estimates of various sketching problems in a distributed setting. This framework ensures that the estimates only have a multiplicative error factor. The technique relies on secure multi-party computation and the sketches submitted by each participant are not private and so cannot be released. Further, the results of Choi et al. do not immediately allow for estimating size or weight of the symmetric difference between two sets.

If the sketching matrix H itself is secret and randomly chosen from a distribution over matrices with entries in a finite field, very strong privacy guarantees on the sketch Hx can be obtained, while still allowing $\|x\|_0$ to be estimated from Hx with small error [7]. Blocki et

al. [8] prove that the Johnson-Lindenstrauss transform is in fact differentially private, when keeping the sketch matrix secret. However, the condition that the sketch matrix is secret is a serious limitation for applications such as streaming and distributed cardinality estimation that require H to be stored or shared.

3 Preliminaries

We let $[n] = \{1, 2, \ldots, n\}$ and let $U = [u]$ be the universe that the datasets are taken from.

For a set $A \subseteq U$, we let x_A denote the characteristic vector for A, defined as

$$(x_A)_j = \begin{cases} 1, & j \in A \\ 0, & \text{otherwise} \end{cases}.$$

We write w_A (or w_{x_A}) for the weight vector for input set A such that

$$w_A = x_A \circ w$$

for fixed, public weights $w_j \in (0, 1]$, and $\circ$ denotes the Hadamard product.

For vector $x = (x_1, ..., x_u)$ we define $\|x\|_p = \left(\sum_{j=1}^{u} x_j^p\right)^{1/p}$ as the p-norm of x. For $p = 0$, we define $\|x\|_0 = \sum_{j=1}^{u} \mathbf{1}[x_j \neq 0]$, often called the zero-"norm". F_0 denotes the 0^{th} frequency moment and represents the number of distinct elements in a stream (or a set). Frequency moments are well-known from the streaming literature, see for example [4].

Our sketch Hx_A is comprised of $\log(u)$ "levels", $H_i x_A$ for $i = 0, ..., \log(u) - 1$. We refer to Section 4.1 for a description of these levels. Let n denote the size of the binary vector representation of $H_i x_A$ for each i. Hence, the size of the noisy KOR sketch $Hx_A + \varphi$ is $\tau = n \log u$. Note that n is fixed and depends on the privacy parameter ε and the accuracy parameter β.

Finally, we assume that sets and vectors are stored in a sparse representation, such that we can list the non-zero entries in the input vector x in time $O(\|x\|_0)$.

3.1 Hashing-based subsampling

The sketch matrix H is defined by several hash functions. For simplicity, we assume access to an oracle representing random hash functions, namely, that we can sample a fully random hash function, and it can be evaluated in constant time. We do not store the hash function as part of our sketch, so the space for our sketch does not include space required for storing the hash function. We believe it is possible to replace these hash functions with concrete, efficient hash functions that can be stored in small space while preserving the asymptotic bounds on accuracy, but in order to focus on privacy aspects, we have not pursued this direction. Importantly, the differential privacy of our method holds for any choice of hash function and does not depend on the random oracle assumption.

To ensure that adding two sketches gives a sketch for the symmetric difference, it is necessary that both players sample the same elements for each H_i. To ensure coordinated sampling, we use a hash function, so the same elements from U are sampled by both players. We use the following (standard) subsampling technique: let $\mathcal{S}$ be the family of all fully random hash functions from U into $[0, 1]$. Let $s \sim \mathcal{S}$ uniformly at random. We sample an element j from the input set at level $i = 0, ..., \log(u) - 1$ if and only if $s(j) \in \left(w_j/2^{i+1}, w_j/2^i\right]$. We refer the reader to the survey of Woodruff [47] for more details on subsampling.

3.2 Differential Privacy

Differential privacy is a statistical property of the behavior of a mechanism [15]. The guarantee is that an adversary who observes the output of a differentially private mechanism will only obtain negligible information about the presence or absence of a particular item in the input data. Intuitively, a differentially private mechanism is almost insensitive to the presence or absence of a single element, in the sense that the probability of observing a specific result should be almost the same for any two neighboring sets.

In Definition 3, we define differential privacy formally in terms of databases. In our application, the databases are sets, and thus *neighboring* means that one set is a subset of the other, and their sizes differ by 1.

▶ **Definition 3** (Differential Privacy [15]). *For $\varepsilon \geq 0$, a randomized mechanism $\mathcal{M}$ is said to be ε-differentially private (or purely differentially private) if for any two neighboring databases, S and T – i.e., databases differing in a single entry – and for all $W \subseteq Range(\mathcal{M})$ it holds that*

$$\Pr\Big[\mathcal{M}(S) \in W\Big] \leq e^{\varepsilon} \cdot \Pr\Big[\mathcal{M}(T) \in W\Big].$$

For $\varepsilon \geq 0$ and $\delta \in [0,1]$, a randomized mechanism $\mathcal{M}$ is said to be (ε, δ)-differentially private (or approximately differentially private) if for any two neighboring databases, S and T, and for all $W \subseteq Range(\mathcal{M})$ it holds that

$$\Pr\Big[\mathcal{M}(S) \in W\Big] \leq e^{\varepsilon} \cdot \Pr\Big[\mathcal{M}(T) \in W\Big] + \delta.$$

We show in section 5.1 that our protocol obtains ε-differential privacy.

Our protocol for estimating the weight of the symmetric difference works in the *local* model of differential privacy, where each player adds noise to their own sketch. It uses the general technique of achieving privacy by adding noise according to *sensitivity* of a function [15]. We note that our sketch would *also* work in a model where vectors supplied by the users are combined using a black-box multi-party *secure aggregation* [21, 33]. In this setting, only the sketch for the symmetric difference would be released, and thus, only this sketch would need to be differentially private, meaning that less noise is required.

We can use the Laplace mechanism [15] to get differentially private estimates of the weights of the input sets. These estimates can be used together with an estimate for the weight of the symmetric difference to compute estimates for the union and the intersection of the two input sets with error that is of the same magnitude as the error for estimating the symmetric difference. For more details about differential privacy, we refer the reader to, for example, [17].

4 Techniques

4.1 Sketch Description

In this section, we describe the noisy KOR sketch in detail. The description is self-contained, but we refer the interested reader to [13] for more background on (linear) sketches. As mentioned, our sketch combines the techniques from [29] with hashing-based subsampling to achieve a sketch that is robust against adding noise, as long as we know how much noise was added.

Recall that for input vector $x \in \{0,1\}$ and public weight vector $w \in (0,1]^u$ we simplify notation by defining $w_x := x \circ w$. The goal is to estimate *the weight of* x, $\|w_x\|_1$.

We first give the intuition behind the $n \times u$-matrices H_i, that our sketch H is comprised of: Suppose that we have a rough estimate $\hat{E}$ of $\|w_x\|_1$, accurate within a constant factor. Then we can obtain a more precise estimate by sampling (using a hash function) a fraction $n/\hat{E}$ of the elements, for some parameter n, and computing the sketch from [29] of size n for the sampled elements. This gives an approximation of the number of sampled elements, which in turn gives an approximation of $\|w_x\|_1$ with small relative error. Since we do not know $\|w_x\|_1$ within a constant factor – especially in the setting where we are interested in the size of the symmetric difference – we use hashing-based subsampling to sample each element j from the input set with probability $w_j/2^{i+1}$ for $i = 0, \ldots, \log(u) - 1$. Thus for each i, we sample elements corresponding to approximately a $1/2^{i+1}$ fraction of the weight and compute the sketch from [29] of size n for the sampled elements. For one of these i we are guaranteed to sample approximately a fraction $n/\|w_x\|_1$ of the input weight assuming that $\|w_x\|_1 > n$. For this i, we can obtain a precise estimate of $\|w_x\|_1$ from the sketch.

We now define H_i formally. We first describe the sketch from [29] as a linear sketch over GF(2). Let $\mathcal{F}$ be the family of all hash functions from universe U into $[n]$, and pick $h \sim \mathcal{F}$ uniformly at random. The hash function h uniquely defines an $n \times u$-matrix K, where

$$K_{k,j} = \begin{cases} 1, & \text{if } h(j) = k \\ 0, & \text{otherwise .} \end{cases}$$

We combine this with the following sampling technique:

Let $\mathcal{S}$ be the family of all hash functions from U to $[0, 1]$. Sample $s \sim \mathcal{S}$ uniformly at random. The hash function s defines a $u \times u$-diagonal matrix S_i for each $i = 0, ..., \log(u) - 1$, defined by

$$(S_i)_{j,j} = \begin{cases} 1, & \text{if } s(j) \in \left(w_j/2^{i+1}, w_j/2^i\right] \\ 0, & \text{otherwise .} \end{cases}$$

The matrix-vector product $S_i x$ represents subsample of input vector x, where we sample each element with probability $w_j/2^{i+1}$.

We are finally ready to define H_i as $H_i = KS_i$, which is an $n \times u$-matrix over GF(2). By definition:

$$(H_i)_{k,j} = \begin{cases} 1, & (h(j) = k) \wedge \left(s(j) \in \left(w_j/2^{i+1}, w_j/2^i\right]\right) \\ 0, & \text{otherwise .} \end{cases}$$

The KOR sketch can be represented as an $n \log(u) \times u$-matrix H, formed by stacking $H_1, ..., H_{\log(u)}$.

Let $\mathcal{N}_\varepsilon$ be a distribution over vectors from $\{0, 1\}^{n \log(u)}$, where each entry is 1 independently with probability p. We show in Section 5.2 that it suffices to set $p = 1/(2 + \varepsilon)$. Sample the noise (or *pertubation*) vector $\varphi \sim \mathcal{N}_\varepsilon$ independently and uniformly at random. The *noisy* KOR sketch of x is then computed (over GF(2)) as:

$$Hx + \varphi.$$

4.2 Estimation

Next, we describe how to compute a weight estimate from a sketch $Hx + \varphi$. Let $w_x := x \circ w$ and let φ_i be the restriction of φ to the entries that are added to $H_i x$ when adding φ to Hx.

To compute an estimate for $\|w_x\|_1$, for each $i = 0, ..., \log(u) - 1$ count the number of 1s in $H_i x + \varphi_i$, $Z_i = \|H_i x + \varphi_i\|_0$ and compute the interval:

$$I_i = \begin{cases} [0, u] & \text{if } Z_i \geq (1-\gamma)n/2 \\ \left[2^i n \ln\left(\frac{\frac{1}{2/\varepsilon+1}}{1-\frac{2Z_i}{(1+\gamma)n}} \right), 2^i n \ln\left(\frac{\frac{1}{2/\varepsilon+1}}{1-\frac{2Z_i}{(1-\gamma)n}} \right) \right] & \text{otherwise.} \end{cases} \tag{1}$$

where $\gamma < \frac{\beta - 1/n}{7e^3(2/\varepsilon+1)}$. Compute the intersection $I = \bigcap_{i=0}^{\log(u)-1} I_i$ and check if the maximum value in I is within a factor $(1+\eta)$ of the minimum value in I for

$$\eta = \frac{6\gamma\left(e^3\left(\frac{2}{\varepsilon}+1\right)-1\right)}{1+\gamma-2\gamma\left(e^3\left(\frac{2}{\varepsilon}+1\right)\right)} \ .$$

If that is the case, every element in I is a good estimate for $\|w_x\|_1$ (having relative error at most $(1+\beta)$) with high probability. Otherwise, $\|w_x\|_1$ is small with high probability, and we let the estimate for $\|w_x\|_1$ be 0. We analyze the accuracy of this estimator in Section 5.

4.3 Application to symmetric difference

In this section, we describe a differentially private protocol to compute an estimate for the weight of the symmetric difference between sets held by two parties. First, we show that the sum of two noisy KOR sketches, $Hx_A + \varphi$ and $Hx_B + \psi$, is a noisy KOR sketch for the symmetric difference, $H(x_{A \triangle B}) + (\varphi + \psi)$, which has the same properties as $Hx_A + \varphi$ and $Hx_B + \psi$, but for $\varepsilon' < \varepsilon$ as more noise is added.

▶ **Lemma 4.** *The addition (over $GF(2)$) of two noisy KOR sketches with perturbation vectors $\varphi \sim \mathcal{N}_\varepsilon$ and $\psi \sim \mathcal{N}_\varepsilon$, respectively, is a noisy KOR sketch for the symmetric difference of the input sets with noise $\varphi + \psi \sim \mathcal{N}_{\varepsilon'}$ for $\varepsilon' = \varepsilon^2/(2+2\varepsilon)$.*

Proof. Let x_A and x_B be the input vectors from each of the two players. Let H be as defined in Section 4.1, and define φ, ψ as the noise vectors for the noisy KOR sketches for x_A and x_B, respectively. We have (over GF(2)) that

$$\begin{aligned} (Hx_A + \varphi) + (Hx_B + \psi) &= (Hx_A + Hx_B) + (\varphi + \psi) \\ &= H(x_A + x_B) + (\varphi + \psi) \, . \end{aligned}$$

This is exactly the noisy KOR sketch for the symmetric difference with perturbation $\varphi + \psi$. Note that we observe a 1 in an entry of $\varphi + \psi$ with probability $p' = p(1-p) + (1-p)p = 2p(1-p)$. We show in Section 5.2 that we can let $p = \frac{1}{2+\varepsilon}$. Observe that

$$p' = \frac{1}{2+\varepsilon'} = \frac{2}{2+\varepsilon}\left(1 - \frac{1}{2+\varepsilon}\right)$$

which implies that $\varepsilon' = \varepsilon^2/(2+2\varepsilon)$. ◀

By Lemma 4 we can treat a sketch for the symmetric difference exactly like a sketch for input vector x although with a different privacy parameter ε'. Hence, Theorem 1 gives us Corollary 2, restated here for convenience:

▶ **Corollary 2.** *For accuracy parameter $\beta > 0$, consider an ε-differentially private noisy KOR sketch for a set A and an ε-differentially private noisy KOR sketch for a set B, based on the same linear sketch $H \sim \mathcal{H}$, sampled independently of A and B. We can compute an approximation $\hat{\Delta}$ of the weight of the symmetric difference, such that with probability $1 - 1/u$:*

$$|\|w_{A \triangle B}\|_1 - \hat{\Delta}| < \beta \|w_{A \triangle B}\|_1 + \text{poly}(1/\varepsilon, 1/\beta, \log u) \ .$$

Note that the additive error in Corollary 2 still depends polynomially on ε even for privacy parameter ε', which is explained by the fact that $\varepsilon' = \varepsilon^2/(2+2\varepsilon)$.

Finally, we assumed that $\|w_A\|_1$ and $\|w_B\|_1$ were released with Laplacian noise, which gives an expected additive error of $O(1/\varepsilon)$ for each of $\|w_A\|_1$ and $\|w_B\|_1$ [15]. We can use the following equations to get estimates for the union, intersection and difference:

$$\|w_{A\cup B}\|_1 = \frac{\|w_A\|_1 + \|w_B\|_1 + \|w_{A\triangle B}\|_1}{2},$$
$$\|w_{A\cap B}\|_1 = \frac{\|w_A\|_1 + \|w_B\|_1 - \|w_{A\triangle B}\|_1}{2}$$
$$\|w_{A\setminus B}\|_1 = \frac{\|w_A\|_1 + \|w_{A\triangle B}\|_1 - \|w_B\|_1}{2}.$$

That is, the error is bounded by half the error of the estimate of the symmetric difference size plus $O(1/\varepsilon)$.

5 Proof of Theorem 1

In this section we give a proof of Theorem 1, restated here for convenience:

▶ **Theorem 1** (Noisy KOR sketch). *Let $w \in (0,1]^u$ be given. For every choice of $0 < \beta < 1$ and $\varepsilon = O(1)$ there exists a distribution $\mathcal{H}$ over GF(2)-linear sketches mapping a vector $x \in \{0,1\}^u$ to $\{0,1\}^\tau$, where $\tau = O\left(\log^2(u)\varepsilon^{-2}\beta^{-2}\right)$, and a distribution $\mathcal{N}_\varepsilon$ over noise vectors such that:*

1. *For $H \sim \mathcal{H}$ and $\varphi \sim \mathcal{N}_\varepsilon$, given $Hx+\varphi$ we can compute, in time $O(\tau)$, an estimate $\hat{w}$ of $\|x\circ w\|_1$ that with probability $1-1/u$ satisfies $|\hat{w} - \|x\circ w\|_1| < \beta\|x\circ w\|_1 + O\left(\log(u)\varepsilon^{-2}\beta^{-2}\right)$.*
2. *For every H in the support of $\mathcal{H}$, $Hx+\varphi$ is ε-differentially private over the choice of $\varphi \sim \mathcal{N}_\varepsilon$, and can be computed in time $O(\|x\|_0 \log(u) + \tau)$, including time for sampling φ.*

5.1 Noise level and Differential Privacy Guarantees

We first show that the noisy KOR sketch $Hx+\varphi$ satisfies ε-differential privacy, which proves part 2 of Theorem 1. Intuitively, removal/insertion of a single element can change only a single entry in the sketch, as the element is inserted into only a single level.

▶ **Lemma 5.** *If $p \in \left(\frac{1}{e^\varepsilon+1}, \frac{1}{2}\right)$ then $Hx+\varphi$ is ε-differentially private.*

Proof. Let A and B be neighboring input sets with corresponding characteristic vectors, x_A and x_B. *Neighboring* means that one set is a subset of the other and the sizes differ by 1. By symmetry of differential privacy, we can without loss of generality assume that A is the smaller set. Suppose that $B\setminus\{z\} = A$. The element z can only affect $H_i x$ for i where z is sampled. Recall that there is at most one such i. If z is never sampled, then $Hx_A = Hx_B$ and privacy is trivial. So assume $i \in \{0,...,\log(u)-1\}$ such that $s(z) \in \left(w_z/2^{i+1}, w_z/2^i\right]$. We limit our attention to $H_i x_A + \varphi_i$, where we think of φ_i as the restriction of the $n\log(u)$-dimensional random vector $\varphi \sim \mathcal{N}_\varepsilon$ to the entries that would be added to $H_i x_A$ when adding φ to Hx_A. We show that $H_i x_A + \varphi_i$ is ε-differentially private. Then the entire sketch, $Hx_A + \varphi$, is ε-differentially private.

Inserting z into the sketch implies that $H_i x_A$ and $H_i x_B$ will differ in exactly one entry, i.e., $\|H_i x_A + H_i x_B\|_0 = 1$. Fix a noisy sketch, S_i. There exist unique vectors φ_i and ψ_i, such that $S_i = H_i x_A + \varphi_i = H_i x_B + \psi_i$. Note that $\|\varphi_i - \psi_i\|_0 = 1$. Let $\|\varphi_i\|_0 = r$. Then

$\|\psi_i\|_0 = r'$ for $r' \in \{r+1, r-1\}$. Conditioned on $\|\varphi_i\|_0 = r$ and $\|\psi_i\|_0 = r'$, the probabilities of randomly drawing exactly these randomness vectors are, respectively:

$$(1-p)^{n-r}p^r \qquad \text{and} \qquad (1-p)^{n-r'}p^{r'}.$$

Let $\varepsilon = O(1)$ be given. By Section 3.2 it is enough to show that for any fixed output $S_i = H_i x_A + \varphi_i = H_i x_B + \psi_i$, we have

$$e^{-\varepsilon} \leq \frac{\Pr\left[\text{observe } S_i \text{ from } A\right]}{\Pr\left[\text{observe } S_i \text{ from } B\right]} = \frac{\Pr\left[\text{observe } H_i x_A + \varphi_i \text{ from } A\right]}{\Pr\left[\text{observe } H_i x_B + \psi_i \text{ from } B\right]} \leq e^{\varepsilon}.$$

where the probability is over the randomness in φ_i and ψ_i. The sketches for A and B are computed using the same H_i, so the choice of H_i has no impact.

Hence, to obtain differential privacy it suffices that for every possible value of r and $r' \in \{r+1, r-1\}$

$$e^{-\varepsilon} \leq \frac{(1-p)^{n-r}p^r}{(1-p)^{n-r'}p^{r'}} = \frac{1}{(1-p)^{r-r'}p^{r'-r}} \leq e^{\varepsilon},$$

which is satisfied for $1/2 > p \geq 1/(e^{\varepsilon}+1)$, since $p < 1/2$ by assumption. ◀

5.2 Bounding accuracy

In this section, along with Section 5.3, we prove the first part of Theorem 1. Let input vector x be given and let $w_x = x \circ w$. We will mainly consider each $H_i x$ isolated, so let the (binary) randomness vector φ_i be the n-dimensional restriction of φ as described in the proof of Lemma 5. First, we state two useful lemmas.

▶ **Lemma 6.** *For each $i = 0, ..., \log(u) - 1$ let $L_i = \|H_i x\|_0$ and $Z_i = \|H_i x + \varphi_i\|_0$. Then:*

$$\mathop{\mathrm{E}}_{\substack{h\sim\mathcal{F},\\ s\sim\mathcal{S}}}[L_i] = \frac{n}{2}\left(1 - \prod_{j:x_j=1}\left(1 - \frac{w_j}{2^i n}\right)\right) \tag{2}$$

$$\mathop{\mathrm{E}}_{\substack{h\sim\mathcal{F},\\ s\sim\mathcal{S},\\ \varphi_i\sim\mathcal{N}_p}}[Z_i] = \frac{n}{2}\left(1 - (1-2p)\prod_{j:x_j=1}\left(1 - \frac{w_j}{2^i n}\right)\right) \tag{3}$$

Proof. We refer the reader to the full version of the paper for the proof. ◀

▶ **Lemma 7.** *For $i = 0, ..., \log(u) - 1$ let $Z_i = \|H_i x + \varphi_i\|_0$. For any $0 < \gamma < 1$, we have with probability at least $1 - 6\log(u)e^{-\frac{\gamma^2 p^3 n}{6^2 \cdot 3}}$ that for all $i = 0, ..., \log(u) - 1$ simultaneously:*

$$(1-\gamma)\mathop{\mathrm{E}}_{\substack{h\sim\mathcal{F},\\ s\sim\mathcal{S},\\ \varphi_i\sim\mathcal{N}_p}}[Z_i] < Z_i < (1+\gamma)\mathop{\mathrm{E}}_{\substack{h\sim\mathcal{F},\\ s\sim\mathcal{S},\\ \varphi_i\sim\mathcal{N}_p}}[Z_i].$$

Proof. We refer the reader to the full version of the paper for the proof. ◀

First, we consider the case when $1 < n < \|w_x\|_1$. In Lemma 8 we state that in this case, with high probability we get an error of at most a factor $(1+\beta)$ for a well-chosen γ, where γ is a function of the privacy parameter ε, the accuracy parameter β and the size of the universe, u. For convenience, define

$$I_i(p) = \begin{cases} [0, u] & \text{if } Z_i \geq (1-\gamma)n/2 \\ \left[2^i n \ln\left(\frac{1-2p}{1-\frac{2Z_i}{(1+\gamma)n}}\right), 2^i n \ln\left(\frac{1-2p}{1-\frac{2Z_i}{(1-\gamma)n}}\right)\right] & \text{otherwise} \end{cases} \tag{4}$$

and $\hat{w}_x := 2^i n \ln\left(1/\prod_{j:x_j=1}\left(1-\frac{w_j}{2^i n}\right)\right)$. We prove our result in two steps:

1. If $\hat{w}_x \in I_i(p)$ for all $i = 0, ..., \log(u)-1$, then there is some i such that any value from (4) estimates $\hat{w}_x$ up to a factor $(1+\eta)$, where η is a function of γ and ε.
2. $\|w_x\|_1 \leq \hat{w}_x \leq \left(1+\frac{1}{2^i n}\right)\|w_x\|_1$ for each i. Specifically, $\|w_x\|_1 \leq \hat{w}_x \leq \left(1+\frac{1}{n}\right)\|w_x\|_1$ for all i.

Hence, we choose γ independent of i such that $(1+\eta)\left(1+\frac{1}{n}\right) \leq (1+\beta)$ for at least one of the intervals $I_i(p)$. We pick γ to work for the i where $\|w_x\|_1/(2^i n) \in [1,2)$ as this corresponds to having an input of size between n and $2n$ (we obtain this input size by the sampling from x in H_i). If $\|w_x\|_1 \geq n$, there is such an i, and we can identify it by checking that the endpoints of the interval are sufficiently close together, as described in Section 4.2. We consider the case when $\|w_x\|_1 < n$ in Section 5.3 where we show that in this case, the error is bounded by an additive factor of $O(n)$.

▶ **Lemma 8.** *Assume $\|w_x\|_1 > n > 1$, and $\beta > \frac{1}{n}$. With probability at least $1-6\log(u)e^{-\frac{\gamma^2 p^3 n}{108}}$ there exists an $i \in \{0, ..., \log(u)-1\}$ such that any element from $I_i(p)$ is a $(1+\beta)$-approximation to $\|w_x\|_1$ for*

$$\gamma < \frac{\left(\beta - \frac{1}{n}\right)(1-2p)}{7e^3} \quad .$$

Specifically, i where $\frac{\|w_x\|_1}{2^i n} \in [1,2)$, gives these guarantees.

Proof Sketch. We give an informal sketch of the proof and refer the reader to the full version of the paper for the formal proof. We first remark that for γ as described, Lemma 7 implies that if $\|w_x\|_1/(2^i n) \leq 2$, then $Z_i < (1-\gamma)n/2$ with high probability. Hence, it suffices to consider the intervals from (4) of the form $I_i(p) = \left[2^i n \ln\left(\frac{1-2p}{1-\frac{2Z_i}{(1+\gamma)n}}\right), 2^i n \ln\left(\frac{1-2p}{1-\frac{2Z_i}{(1-\gamma)n}}\right)\right]$. Define

$$\hat{w}_x := 2^i n \ln\left(\frac{1}{\prod_{j:x_j=1}\left(1-\frac{w_j}{2^i n}\right)}\right).$$

From Lemma 6, we have

$$\prod_{j:x_j=1}\left(1-\frac{w_j}{2^i n}\right) = \frac{1-\frac{2\,\mathrm{E}[Z_i]}{n}}{1-2p}.$$

Assume that the bounds in Lemma 7 are satisfied. We remove this assumption shortly. By the bounds in Lemma 7, $\hat{w}_x \in I_i(p)$ for all i. We show that $\hat{w}_x$ is contained in an interval, which is slightly bigger than $I_i(p)$ whenever $\|w_x\|_1/(2^i n) \in [1,2)$ and show that the endpoints of this interval are within a factor $(1+\eta)$ of each other, where η is a function of γ. Clearly, then $I_i(p)$ is also sufficiently small for this i. Denote this interval $I_i^*(p)$. Any element from $I_i^*(p)$

is a $(1+\eta)$-approximation to $\hat{w}_x$. Removing the assumption that the bounds in Lemma 7 hold, we simply get a small error probability and conclude that with probability at least $1-6\log(u)e^{-\gamma^2 p^3 n/108}$ we have $\hat{w}_x \in I_i(p)$ for all i, and thus any value from $I_i^*(p)$ is a $(1+\eta)$ estimation to $\hat{w}_x$ with high probability. Observing that $\|w_x\|_1 < \hat{w}_x \leq \left(1+\frac{1}{n}\right)\|w_x\|_1$ for any i, we choose γ in terms of β such that $(1+\eta)\left(1+\frac{1}{n}\right) < (1+\beta)$. Then any value from $I_i^*(p)$ is a $(1+\beta)$-approximation for $\|w_x\|_1$. We formally choose γ in the full version of the paper. We remark that the assumption $\|w_x\|_1/(2^i n) \in [1,2)$ allows us to choose γ independent of i, such that we can compute $I_i(p)$ for all i with a single value of γ. ◀

Observing that $\frac{1}{2+\varepsilon} > \frac{1}{e^\varepsilon+1}$ for $\varepsilon > 0$, we let $p = 1/(2+\varepsilon)$ and observe that for $I_i := I_i(1/(2+\varepsilon))$ with the choice of γ described in Lemma 8, we get the interval I_i in (1).

5.3 Putting things together

In this section we consider the accuracy in the remaining case where $\|w_x\|_1 \leq n$. We also analyze the running time. Combining with Section 5.1, this completes the proof of Theorem 1.

Note that if $\varepsilon > 1$, we can start our protocol by dividing ε by a suitable constant, c such that $\varepsilon' = \varepsilon/c < 1$. Changing ε by a constant will change our bounds by a constant factor as well. Hence, we can without loss of generality assume $\varepsilon < 1$. We can also, without loss of generality, assume $u > 10$, as this will at most increase the failure probability and space by a constant factor.

We first show a sufficient upper bound on the sketch size $\tau = n\log u$. Observe that $p > 1/4$ and let $c_\gamma = 7e^3$ be a constant. Then we want $e^{-\frac{\gamma^2 p^3 n}{108}} < 1/u^2$ as this ensures a failure probability of at most $6\log(u)/u^2 < 1/u$. Noting that

$$(1-2p)^2 = \left(1-\frac{2}{2+\varepsilon}\right)^2 = \left(\frac{1}{2/\varepsilon+1}\right)^2 = \frac{1}{4/\varepsilon^2+4/\varepsilon+1} > \frac{\varepsilon^2}{20},$$

we have

$$\begin{aligned} e^{-\frac{\gamma^2 p^3 n}{108}} &< e^{-\frac{\left(\frac{\left(\beta-\frac{1}{n}\right)(1-2p)}{7e^3}\right)^2 n/4^3}{108}} = e^{-\frac{\left(\beta-\frac{1}{n}\right)^2\left(\frac{1}{(2/\varepsilon+1)^2}\right)n}{4^3\cdot c_\gamma^2\cdot 108}} \\ &< e^{-\frac{\left(\beta-\frac{1}{n}\right)^2\varepsilon^2 n}{20\cdot 4^3 c_\gamma^2\cdot 108}} < 1/u^2 \end{aligned}$$

when letting $n = O\left(\log(u)\beta^{-2}\varepsilon^{-2}\right)$. Hence, the size of the sketch is

$$\tau = \log(u)\cdot n = O\left(\frac{\log^2(u)}{\varepsilon^2\beta^2}\right).$$

Note that this n satisfies the requirement $\beta > 1/n$ from Lemma 8.

We argue about the error: Note that if $\|w_x\|_1 \geq n$, then if one of the intervals I_i is sufficiently small and $\hat{w}_x \in I_i$ for all $i = 0,\ldots,\log(u)-1$, then $\hat{w}_x \in I = \bigcap_{i=0}^{\log(u)-1} I_i$ and I is also sufficiently small to give the wanted estimate. So by Lemma 8, we can check if the endpoints of I are within a factor at most $(1+\eta)$ of each other, and if so, with probability $1-1/u$ any value from I is within a factor $(1+\beta)$ of $\|w_x\|_1$. If I is too big, then none of the intervals I_i was sufficiently small implying that our assumption that $\|w_x\|_1/(2^i n) \in [1,2)$ does not hold for any i. Hence, with probability $1-1/u$ we have $\|w_x\|_1 < n$. We refer to the formal proof of Lemma 8 in the full version of the paper for the details. Our protocol sets the estimate of $\|w_x\|_1$ to 0 leading to an additive error of $O(n)$ when I was too big. This means that we get an additive error of at most $n = O\left(\log(u)\beta^{-2}\varepsilon^{-2}\right)$, as required.

Finally, we comment on the running times: For the first part of Theorem 1, we note that in order to compute the estimate, we need to count the number of ones in $H_i x + \varphi_i$ for each $i = 0, ..., \log(u) - 1$, compute the intervals I_i and their intersection and check if it is sufficiently small. Counting the number of ones in all $H_i x + \varphi_i$ is the bottleneck and requires time $O(\tau)$. For the second part of Theorem 1, note that we can initialize the randomness vector φ in time $O(\tau)$ and we can hash vector x in time $O(\|x\|_0 \log(u))$ assuming that we can iterate over x in time $O(\|x\|_0)$.

Combining with Lemma 8 and Lemma 5, we have completed the proof of Theorem 1.

6 Distributed Streaming Implementation

In a streaming setting we want a sketch which can be updated and two sketches can be merged to give a sketch for the union of the input streams, while we cannot guarantee that there are no duplicates in the input stream. In this case, our sketch does not immediately apply, as items with an even number of occurrences would "cancel out". Such items would therefore never be represented in the sketch, as the sketch is over GF(2). This issue can easily be fixed: the idea is to add another layer of sampling, such that we sample each *occurrence* of a data item with probability 1/2. Hence, we treat identical items independently on each occurrence and so ensures that an entry in the sketch is 1 with probability 1/2, regardless of the number of copies of identical items and collisions with other items. We refer to this as the *pre-sampled* sketch. The intuition is that the number of copies of an item inserted in the pre-sampled sketch is even or odd with probability 1/2. By Chernoff bounds the fraction of elements that are sampled an odd number of times is very close to 1/2 with high probability. Thus it is natural to consider the estimator that is two times the estimator described in Section 4.2.

To understand this in more detail we argue that merging two (non-private) pre-sampled sketches over $GF(2)$ gives a sketch for the *union* of the two input sets. Suppose $z \in A \cup B$, $h(z) = k$ and that z is sampled at level i. We argue that $\Pr[(H_i x_{A\cup B})_k = 1] = 1/2$. Note that

$$(H_i x_{A\cup B})_k = 1 \qquad \Leftrightarrow \qquad (H_i x_A)_k \neq (H_i x_B)_k.$$

Further, we have that if $z \in A$, then $\Pr[(H_i x_A)_k = 1] = 1/2$ regardless of the number of other elements hashing to k at level i. If no elements from A hash to entry k at level i, then $\Pr[(H_i x_A)_k = 1] = 0$. We have

$$\begin{aligned}\Pr[(H_i x_{A\cup B})_k = 1] = \Pr[(H_i x_A)_k = 1] \Pr[(H_i x_B)_k = 0] \\ + \Pr[(H_i x_A)_k = 0] \Pr[(H_i x_B)_k = 1],\end{aligned}$$

which is 1/2 whenever $z \in A \cup B$.

References

1 Mohammad Alaggan, Sébastien Gambs, and Anne-Marie Kermarrec. BLIP: non-interactive differentially-private similarity computation on bloom filters. In *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS*, pages 202–216, 2012. doi:10.1007/978-3-642-33536-5_20.

2 Mohammad Alaggan, Sébastien Gambs, Stan Matwin, and Mohammed Tuhin. Sanitization of call detail records via differentially-private bloom filters. In *Data and Applications Security and Privacy XXIX - 29th Annual IFIP WG 11.3 Working Conference, DBSec 2015*, pages 223–230, 2015. doi:10.1007/978-3-319-20810-7_15.

3 Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. *Journal of Computer and System Sciences*, 64(3):719–747, 2002.

4 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Symposium on the Theory of Computing*, pages 20–29, 1996. doi:10.1145/237814.237823.

5 Pranjal Awasthi, Maria-Florina Balcan, Nika Haghtalab, and Hongyang Zhang. Learning and 1-bit compressed sensing under asymmetric noise. In *Conference on Learning Theory*, pages 152–192, 2016.

6 Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10, 2002.

7 Valerio Bioglio, Tiziano Bianchi, and Enrico Magli. Secure compressed sensing over finite fields. In *International Workshop on Information Forensics and Security (WIFS)*, pages 191–196, 2014.

8 Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. The johnson-lindenstrauss transform itself preserves differential privacy. In *Symposium on Foundations of Computer Science, FOCS*, pages 410–419, 2012. doi:10.1109/FOCS.2012.67.

9 Andrei Z. Broder and Michael Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2003. doi:10.1080/15427951.2004.10129096.

10 Clément Canonne, Gautam Kamath, and Thomas Steinke. The discrete gaussian for differential privacy. *arXiv preprint arXiv:2004.00010*, 2020.

11 Seung Geol Choi, Dana Dachman-Soled, Mukul Kulkarni, and Arkady Yerukhimovich. Differentially-private multi-party sketching for large-scale statistics. *IACR Cryptol. ePrint Arch.*, 2020:29, 2020. URL: https://eprint.iacr.org/2020/029.

12 Reuven Cohen, Liran Katzir, and Aviv Yehezkel. A unified scheme for generalizing cardinality estimators to sum aggregation. *Information Processing Letters*, 115(2):336–342, 2015.

13 Graham Cormode, Minos N. Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012. doi:10.1561/1900000004.

14 Damien Desfontaines, Andreas Lochbihler, and David A. Basin. Cardinality estimators do not preserve privacy. *PoPETs*, 2019(2):26–46, 2019. doi:10.2478/popets-2019-0018.

15 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *3rd Theory of Cryptography Conference, TCC*, pages 265–284, 2006. doi:10.1007/11681878_14.

16 Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

17 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014. doi:10.1561/0400000042.

18 Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 Conference on Computer and Communications Security*, pages 1054–1067, 2014. doi:10.1145/2660267.2660348.

19 Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *AofA: Analysis of Algorithms*, pages 137–156, 2007.

20 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.

21 Slawomir Goryczka, Li Xiong, and Vaidy S. Sunderam. Secure multiparty aggregation with differential privacy: a comparative study. In *Joint 2013 EDBT/ICDT Conferences, EDBT/ICDT '13*, pages 155–163, 2013. doi:10.1145/2457317.2457343.

22 Peter J Haas, Jeffrey F Naughton, S Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *VLDB*, volume 95, pages 311–322, 1995.

23 Moritz Hardt and Kunal Talwar. On the geometry of differential privacy. In *Symposium on Theory of Computing, STOC*, pages 705–714, 2010. doi:10.1145/1806689.1806786.

24 T. S. Jayram and David P. Woodruff. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *Transactions on Algorithms*, 9(3):26:1–26:17, 2013. doi:10.1145/2483699.2483706.

25 Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the 29th ACM symposium on Principles of database systems (PODS)*, pages 41–52, 2010.

26 Krishnaram Kenthapadi, Aleksandra Korolova, Ilya Mironov, and Nina Mishra. Privacy via the Johnson-Lindenstrauss transform. *J. Priv. Confidentiality*, 5(1), 2013. doi:10.29012/jpc.v5i1.625.

27 Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *VLDB*, volume 4, pages 180–191. Toronto, Canada, 2004.

28 Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of ACM International Conference on Management of data (SIGMOD)*, pages 193–204, 2011.

29 Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Symposium on the Theory of Computing*, pages 614–623, 1998. doi:10.1145/276698.276877.

30 Andrew McGregor, Ilya Mironov, Toniann Pitassi, Omer Reingold, Kunal Talwar, and Salil Vadhan. The limits of two-party differential privacy. In *51st Annual Symposium on Foundations of Computer Science*, pages 81–90, 2010.

31 Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, volume 7, pages 94–103, 2007.

32 Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of ACM International Conference on Management of data (SIGMOD)*, pages 19–30, 2009.

33 Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. In *23rd Annual Network and Distributed System Security Symposium, NDSS*, 2016. doi:10.14722/ndss.2016.23175.

34 Darakhshan Mir, S Muthukrishnan, Aleksandar Nikolov, and Rebecca N Wright. Pan-private algorithms: When memory does not help. *arXiv preprint arXiv:1009.1544*, 2010.

35 Darakhshan Mir, Shan Muthukrishnan, Aleksandar Nikolov, and Rebecca N Wright. Pan-private algorithms via statistics on sketches. In *Proceedings of the 30th Symposium on Principles of Database Systems (PODS)*, pages 37–48, 2011.

36 Ilya Mironov. On significance of the least significant bits for differential privacy. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *Conference on Computer and Communications Security, CCS*, pages 650–661, 2012. doi:10.1145/2382196.2382264.

37 Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil P. Vadhan. Computational differential privacy. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 126–142, 2009. doi:10.1007/978-3-642-03356-8_8.

38 Michael Mitzenmacher, Rasmus Pagh, and Ninh Pham. Efficient estimation for high similarities using odd sketches. In *Proceedings of 23rd international conference on World Wide Web (WWW)*, pages 109–118, 2014.

39 Aleksandar Nikolov. Personal communication. Clarification, 2020.

40 Hagen Sparka, Florian Tschorsch, and Björn Scheuermann. P2KMV: A privacy-preserving counting sketch for efficient and accurate set intersection cardinality estimations. *IACR Cryptology ePrint Archive*, 2018:234, 2018. URL: http://eprint.iacr.org/2018/234.

41 Rade Stanojevic, Mohamed Nabeel, and Ting Yu. Distributed cardinality estimation of set operations with differential privacy. In *IEEE Symposium on Privacy-Aware Computing, PAC*, pages 37–48, 2017. doi:10.1109/PAC.2017.43.

42 Florian Tschorsch and Björn Scheuermann. An algorithm for privacy-preserving distributed user statistics. *Computer Networks*, 57(14):2775–2787, 2013. doi:10.1016/j.comnet.2013.05.011.

43 Salil P. Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer, 2017. doi:10.1007/978-3-319-57048-8_7.

44 Saskia Nuñez von Voigt and Florian Tschorsch. Rrtxfm: Probabilistic counting for differentially private statistics. In *Workshop on Trust and Privacy Aspects of Smart Information Environments (TPSIE)*, 2019.

45 Stanley L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965. URL: http://www.jstor.org/stable/2283137.

46 Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private SQL with bounded user contribution. *Proceedings on Privacy Enhancing Technologies*, 2020(2):230–250, 2020.

47 David P. Woodruff. Data streams and applications in computer science. *Bulletin of the EATCS*, 114, 2014. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/304.

Fine-Grained Complexity of Regular Path Queries

Katrin Casel
Hasso Plattner Institute, Universität Potsdam, Germany

Markus L. Schmid
Humboldt-Universität zu Berlin, Germany

Abstract

A regular path query (RPQ) is a regular expression q that returns all node pairs (u, v) from a graph database that are connected by an arbitrary path labelled with a word from $L(q)$. The obvious algorithmic approach to RPQ evaluation (called PG-approach), i. e., constructing the product graph between an NFA for q and the graph database, is appealing due to its simplicity and also leads to efficient algorithms. However, it is unclear whether the PG-approach is optimal. We address this question by thoroughly investigating which upper complexity bounds can be achieved by the PG-approach, and we complement these with conditional lower bounds (in the sense of the fine-grained complexity framework). A special focus is put on enumeration and delay bounds, as well as the data complexity perspective. A main insight is that we can achieve optimal (or near optimal) algorithms with the PG-approach, but the delay for enumeration is rather high (linear in the database). We explore three successful approaches towards enumeration with sub-linear delay: super-linear preprocessing, approximations of the solution sets, and restricted classes of RPQs.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Problems, reductions and completeness; Theory of computation → Database query languages (principles); Theory of computation → Data structures and algorithms for data management

Keywords and phrases Graph Databases, Regular Path Queries, Enumeration, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.19

Related Version *Full Version*: `https://arxiv.org/abs/2101.01945`

Funding *Katrin Casel*: Supported by the Federal Ministry of Education and Research of Germany (BMBF) in the KI-LAB-ITSE framework – project number 01IS19066.
Markus L. Schmid: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 416776735 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 416776735).

Acknowledgements We wish to thank the anonymous reviewers for their valuable feedback. In particular, following the reviewer's comments and suggestions, we have included more background information and comprehensive explanations of certain aspects, which substantially improved the overall exposition of this paper.

1 Introduction

An essential component of graph query languages (to be found both in academical prototypes as well as in industrial solutions) are *regular path queries* (RPQs). Abstractly speaking, a regular expression q over some alphabet Σ is interpreted as query that returns from a Σ-edge-labelled, directed graph $\mathcal{D}$ (i. e., a *graph database*) the set $q(\mathcal{D})$ of all node pairs (u, v) that are connected by a q-path, i. e., a path labelled with a word from q's language (and possibly also a witness path per node pair, or even all such paths). This simple, yet relevant concept has heavily been studied in database theory (the following list is somewhat biased towards recent work): results on RPQs [26, 7, 35, 37, 40, 16], conjunctive RPQs [15, 43, 9] and extensions thereof [34, 12, 34, 29], questions of static analysis [28, 11, 27, 31, 44], experimental analyses [17, 19, 39], and surveys of this research area [10, 53, 5, 23].

In the simplest setting, where we are only interested in the node pairs (but no paths) connected by *arbitrary* q-paths (instead of, e. g., simple paths), evaluation can be done efficiently. Deviating from this simple setting, however, leads to intractability: if we ask for nodes connected by *simple paths* (no repeated nodes), or connected by *trails* (no repeated arcs), then RPQ evaluation is NP-hard even in data-complexity (see [41, 7, 38] and [37], respectively). Note that the simple path and trail semantics are mostly motivated by the fact that under these semantics there is only a finite number of q-paths per node pair. If we move to conjunctions of RPQs (CRPQs) or even more powerful extensions motivated by practical requirements, then also with the arbitrary path semantics evaluation becomes intractable in combined complexity (i. e., they inherit hardness from relational conjunctive queries (CQs)).

In order to guide practical developments in the area of graph databases, the computational hard cases of RPQ (and CRPQ) evaluation have been thoroughly investigated in database theory. However, with respect to arbitrary q-paths, research seems to have stopped at the conclusion that efficient evaluation is possible by the following simple *PG-approach*: given graph database $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$ and NFA M_q for q with state set V_q, construct the *product graph* $G_{\mathcal{D},q}$ with nodes $V_\mathcal{D} \times V_q$ and an arc $((u,p),(v,p'))$ iff, for some $\mathtt{a} \in \Sigma$, $\mathcal{D}$ has an arc $(u,\mathtt{a},v)$ and M_q has an arc $(p,\mathtt{a},p')$, and then use simple graph-searching techniques on $G_{\mathcal{D},q}$.

The PG-approach is explicitly defined in several papers, e. g., [41, 10, 38], and mainly used to prove a worst-case upper bound (actually $\mathrm{O}(|q||\mathcal{D}|)$ for Boolean evaluation); in [38] it is used for enumerating q-paths between two given nodes. But it is also very appealing from a practical point of view due to its simplicity: we are just coupling well-understood algorithmic concepts like finite automata and graph reachability algorithms. Arguably, implementing the PG-approach is an exercise suitable for a first year programming course (making it feasible and cost-efficient for industrial systems). As it seems, putting RPQ evaluation with arbitrary path semantics and the respective PG-approach into the focus of a thorough theoretical study has not yet been done. This paper is devoted to this task. In particular, we wish to investigate the following two (somewhat overlapping) aspects:

1. Applicability of the PG-approach: the PG-approach is suited for solving simple evaluation problems like checking $q(\mathcal{D}) = \emptyset$ or $(u,v) \in q(\mathcal{D})$ (for given $u, v \in V_\mathcal{D}$), but is it also appropriate for more relevant tasks like computing, counting or enumerating $q(\mathcal{D})$?
2. Optimality of the PG-approach: Does the PG-approach lead to optimal algorithms, or can it be beaten by conceptionally different techniques?

Answering these questions provides a better theoretical understanding of RPQ-evaluation (which, as mentioned above, are at the heart of many graph query languages). But also for the more powerful CRPQs and more practically motivated graph query languages, we can derive valuable insights from our investigation. Let us mention two such examples (a complete summary of our results follows further down). As noted in [10], we can reduce CRPQ evaluation to the evaluation of relational CQs by first constructing all tables represented by the single RPQs and then evaluating a CQ over this database. To do this, we first have to compute the results of all RPQs, so it seems helpful to know the best algorithms for this intermediate task. Moreover, if we want to benefit from the existing CQ evaluation techniques (e. g., exploiting acyclicity etc.) we are more or less forced to this two-step approach. With respect to *enumerations* of CQs, it is known that linear preprocessing and constant delay enumeration is possible provided that the CQs satisfy certain acyclicity properties (see [8, 14], or the surveys [13, 45]). Unfortunately, these techniques do not carry over to CRPQs since, as we show, linear preprocessing and constant delay enumeration is not possible even for single RPQs (conditional to some complexity assumptions).

Table 1 All upper bounds can be achieved as running times of some algorithm, while the lower bounds cannot be achieved as running time by any algorithm, unless the displayed hypothesis fails. The exponent ω denotes the best known matrix multiplication exponent.

Non-enum. Results		Boole, Test, Witness	Eval	Count
upper bounds		$O(\|\mathcal{D}\|\|q\|)$	$O(\|V_\mathcal{D}\|\|\mathcal{D}\|\|q\|)$ $O((\|V_\mathcal{D}\|\|q\|)^{2.37})$	$O(\|V_\mathcal{D}\|\|\mathcal{D}\|\|q\|)$ $O((\|V_\mathcal{D}\|\|q\|)^{\omega})$
lower bounds	OV & com-BMM	$O((\|\mathcal{D}\|\|q\|)^{1-\epsilon})$	—	—
	OV	—	—	$O^{dc}((\|V_\mathcal{D}\|\|\mathcal{D}\|)^{1-\epsilon})$
	SBMM	—	$O^{dc}(\|q(\mathcal{D})\|+\|\mathcal{D}\|)$	—
	com-BMM	—	$O^{dc}((\|V_\mathcal{D}\|\|\mathcal{D}\|)^{1-\epsilon})$	—

Since the problem we investigate can be solved in polynomial time (also in combined complexity), we cannot show lower bounds in terms of hardness results for complexity classes like NP or PSPACE. Instead, we make use of the framework of *fine-grained complexity*, which allows to prove lower bounds that are conditional on some algorithmic assumptions (see the surveys [50, 22, 51]). In particular, fine-grained complexity is a rather successful toolbox for giving evidence that the *obvious* algorithmic approach to some basic problem, is also the *optimal* one. This is exactly our setting here, with respect to RPQ-evaluation and the PG-approach. To the knowledge of the authors, such conditional lower bounds are not yet a well-established technique in database theory (however, see Section [13, Section 6] for a survey of conditional lower bounds in the context of CQ enumeration).

A main challenge is that fine-grained complexity is not exactly tailored to either the data-complexity perspective or to enumeration problems. We will next outline our results.

1.1 Our Contribution

All investigated RPQ-evaluation problems are summarised on page 7. In the following, $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$ is the graph database, q is the RPQ, and $\epsilon > 0$. With the notation $O^{dc}(\cdot)$, we hide factors $f(|q|)$ for some function f (i. e., it is used for stating data-complexities). All lower bounds mentioned in the following are conditional to some of the algorithmic assumptions summarised in Section 4 (we encourage the reader less familiar with fine-grained complexity hypotheses to have a look at this section first, which can be read independently). For presentational reasons, we do not always explicitly mention this in the rest of the introduction and when we say that a certain running time is "not possible", this statements is always conditional in this sense (see Tables 1 and 2 for the actual hypotheses). As common in fine-grained complexity, we rule out *true* sub-linear ($O(n^{1-\epsilon})$), sub-quadratic ($O(n^{2-\epsilon})$), or sub-cubic ($O(n^{3-\epsilon})$) running times, but not possible improvements by logarithmic factors.

Non-Enumeration Variants

The following results are summarised in Table 1. For the simple problems Boole (checking $q(\mathcal{D}) = \emptyset$), Test (checking $(u, v) \in q(\mathcal{D})$) and Witness (computing some element from $q(\mathcal{D})$), the PG-approach yields an upper bound of $O(|\mathcal{D}||q|)$, which is optimal (since linear) in data complexity, and we can show lower bounds demonstrating its optimality also in combined complexity. For Eval (computing the set $q(\mathcal{D})$) the PG-approach yields a data complexity upper bound of $O^{dc}(|V_\mathcal{D}||\mathcal{D}|)$, which cannot be improved by *combinatorial* algorithms, although $O^{dc}(|V_\mathcal{D}|^{2.37})$ is possible by fast matrix multiplication (see Section 4 for a discussion of the meaning of the term "combinatorial"). In addition, we can show that

Table 2 All upper bounds can be achieved as running times of some algorithm, while the lower bounds cannot be achieved as running time by any algorithm, unless the displayed hypothesis fails. The exponent ω denotes the best known matrix multiplication exponent.

Enum. Results		Enum			
		preprocessing	delay	sorted	updates
upper bound		$O(\lvert\mathcal{D}\rvert\lvert q\rvert)$	$O(\lvert\mathcal{D}\rvert\lvert q\rvert)$	✓	$O(1)$
lower bounds	OV & com-BMM	$O(\text{delay})$	$O((\lvert\mathcal{D}\rvert\lvert q\rvert)^{1-\epsilon})$	×	×
	SBMM	$O^{dc}(\lvert\mathcal{D}\rvert)$	$O^{dc}(1)$	×	×
	com-BMM	$O^{dc}(\lvert\mathcal{D}\rvert)$	$O^{dc}(\lvert V_\mathcal{D}\rvert^{1-\epsilon})$	×	×
	OMv	arbitrary	$O^{dc}(\lvert V_\mathcal{D}\rvert^{1-\epsilon})$	×	$O^{dc}(\lvert V_\mathcal{D}\rvert^{1-\epsilon})$
	com-BMM	$O^{dc}(\lvert\mathcal{D}\rvert)$	$O^{dc}(\lvert V_\mathcal{D}\rvert^{2-\epsilon})$	×	$O^{dc}(\lvert V_\mathcal{D}\rvert^{2-\epsilon})$

linear time data complexity, i. e., $O^{dc}(|q(\mathcal{D})|+|\mathcal{D}|)$, is not possible even for non-combinatorial algorithms. For Count (computing $|q(\mathcal{D})|$), we get $O^{dc}(|V_\mathcal{D}||\mathcal{D}|)$ as upper and lower bound, not restricted to combinatorial algorithms.

Enumeration

Our results for RPQ-enumeration are summarised in Table 2. An entry "O(delay)" in column "preprocessing" means that the preprocessing is bounded by the delay (which means that no preprocessing is required). The column "sorted" indicates whether the enumeration is produced lexicographically sorted.

In comparison to the non-enumeration problem variants, the picture is less clear and deserves more explanation. The PG-approach yields a simple enumeration algorithm with delay $O(|\mathcal{D}||q|)$, that also trivially supports updates in constant time, since the preprocessing fits into the delay bound. Our lower bounds for Boole also mean that this delay cannot be improved in terms of *combined complexity*. While this lower bound was interesting for problems like Boole etc., it now gives a correct answer to the wrong question. The main goal now should be to find out whether we can remedy the linear dependency of the delay on $|\mathcal{D}|$, at the expense of spending more time in terms of $|q|$, or of losing the ability of handling updates, or even of allowing a slightly super-linear preprocessing.

In this regard, the strongest result would be linear preprocessing $O(|\mathcal{D}|f(|q|))$ and constant delay $O(f(|q|))$. However, we can rule this out even for algorithms not capable of handling updates. Then, the next question is which non-constant delays can be achieved that are strictly better than linear. For example, none of our lower bounds for the non-enumeration variants suggest that linear preprocessing and a delay bounded by, e. g., $|V_\mathcal{D}|$ or the degree of $\mathcal{D}$, should not be possible. We are *not* able to answer this question in its general form (and believe it to be very challenging), but we are able to provide several noteworthy insights.

For linear preprocessing, a delay of $O(|V_\mathcal{D}|)$ (if possible at all) cannot be beaten by combinatorial algorithms (even without updates). This can be strengthened considerably, if we also require updates in some reasonable time: for general algorithms (i. e., not necessarily combinatorial) delay *and* update time strictly better than $O^{dc}(|V_\mathcal{D}|)$ is not possible even with arbitrary preprocessing, and for combinatorial algorithms with linear preprocessing even delay *and* update time of $O(|\mathcal{D}|)$ cannot be beaten. This last result nicely complements the upper bound at least for combinatorial algorithms and in the dynamic case.

In summary, for linear preprocessing, $O(|V_\mathcal{D}|)$ is a lower bound for the delay and if we can beat $O(|\mathcal{D}|)$, we should not be able to also support updates.

Enumeration of Restricted Variants

Finally, we obtain restricted problem variants that can be solved with delay strictly better than $O(|\mathcal{D}|)$ (in data complexity). We explore three different approaches: (1) by allowing super-linear preprocessing of $O^{dc}(\overline{\Delta}(\mathcal{D})\log(\overline{\Delta}(\mathcal{D}))|\mathcal{D}|)$ (where $\overline{\Delta}(\mathcal{D})$ is the average degree of $\mathcal{D}$), we can achieve a delay of $O(|V_{\mathcal{D}}|)$; (2) in linear preprocessing and constant delay, we can enumerate a representative subset of $q(\mathcal{D})$ instead of the whole set $q(\mathcal{D})$; (3) for a subclass of RPQs, we can solve RPQ-Enum with linear preprocessing and delay $O(\Delta(\mathcal{D}))$ (where $\Delta(\mathcal{D})$ is the maximum degree of $\mathcal{D}$).

Due to space constraints, most proof details are omitted, although we give proof sketches for some results; detailed proofs can be found in the preliminary full version of this paper [25].

2 Main Definitions

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $[n] = \{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$. For a finite alphabet A, A^+ denotes the set of non-empty words over A and $A^* = A^+ \cup \{\varepsilon\}$ (where ε is the empty word). For a word $w \in A^*$, $|w|$ denotes its length;$w^1 = w$ and $w^k = ww^{k-1}$ for every $k \geq 2$. For $L, K \subseteq A^*$, let $L \cdot K = \{w_1 \cdot w_2 \mid w_1 \in L, w_2 \in K\}$, let $L^1 = L$ and $L^k = L \cdot L^{k-1}$ for every $k \geq 2$, let $L^+ = \bigcup_{k \geq 1} L^k$ and $L^* = L^+ \cup \{\varepsilon\}$.

2.1 Σ-Graphs

We now define the central graph model that is used to represent graph databases as well as finite automata. Let Σ be a finite alphabet of constant size. A *Σ-graph* is a directed, edge labelled multigraph $G = (V, E)$, where V is the set of *vertices* (or *nodes*) and $E \subseteq V \times (\Sigma \cup \{\varepsilon\}) \times V$ is the set of *edges* (or *arcs*). For $u \in V$ and $x \in \Sigma \cup \{\varepsilon\}$, $E_x(u) = \{v \mid (u, x, v) \in E\}$ is the set of *x-successors of u*. A path from $w_0 \in V$ to $w_k \in V$ of length $k \geq 0$ is a sequence $p = (w_0, a_1, w_1, a_2, w_2 \ldots, w_{k-1}, a_k, w_k)$ with $(w_{i-1}, a_i, w_i) \in E$ for every $i \in [k]$. We say that p is *labelled* with the *word* $a_1 a_2 \ldots a_k \in \Sigma^*$. According to this definition, for every $v \in V$, (v) is a path from v to v of length 0 that is labelled by ε. Hence, every node v of a Σ-graph has an ε-labelled path to itself, even though there might not be an ε-arc from v to v. Moreover, due to ε as a possible edge-label, paths of length k may be labelled with words w with $|w| < k$. The size of $G = (V, E)$ is $|G| = \max\{|V|, |E|\}$.

For any Σ-graph $G = (V, E)$, we call $(V, \{(u, v) \mid u \neq v \wedge \exists x \in \Sigma \cup \{\varepsilon\} : (u, x, v) \in E\})$ the *underlying graph* of G (note that the underlying graph is simple, non-labelled and has no loops). In particular, by a slight abuse of notation, we denote by E^* the reflexive-transitive closure of the underlying graph of G. Since we always assume $|\Sigma|$ to be a constant, we have that $|G| = \Theta(\max\{|V|, |\{(u, v) \mid u \neq v \wedge \exists x \in \Sigma \cup \{\varepsilon\} : (u, x, v) \in E\}|\})$ (i. e., $|G|$ is asymptotically equal to the size of its underlying graph). For every $u \in V$, the *degree of u* is $\Delta(u) = |\bigcup_{x \in \Sigma \cup \{\varepsilon\}} E_x(u)|$ (so $\Delta(u)$ is actually the out-degree), and the *maximum degree* of G is $\Delta(G) = \max\{|\Delta(u)| \mid u \in V\}$. The *average degree* of G is $\overline{\Delta}(G) = \frac{1}{|V|}\sum_{u \in V} |\Delta(u)|$. Obviously, $\overline{\Delta}(G) \leq \Delta(G) \leq |V|$.

Since Σ-graphs are the central data structures for our algorithms, we have to discuss implementational aspects of Σ-graphs in more detail. The set V of a Σ-graph $G = (V, E)$ is represented as a list, and, for every $u \in V$ and for every $x \in \Sigma \cup \{\varepsilon\}$, we store a list of all x-successors of u, which is called the *x-adjacency list* for u. We assume that we can check in constant time whether a list is empty and we can insert elements in constant time. However, finding and deleting an element from a list requires linear time. Furthermore, we assume that we always store together with a node a pointer to its adjacency list (thus, we can always retrieve the x-adjacency list for a given node in constant time).

▶ Remark 1. All lower bounds presented in this paper hold for any graph representation that can be constructed in time linear in $|G| = \max\{|V|, |E|\}$. For the upper bounds, we chose the simple representation with adjacency lists as it emerged as the natural structure for our enumeration approach; let us point out here that since we always store pointers to the adjacency lists along with the nodes, we can perform a breadth-first search (BFS) from any given start node u in time $\mathrm{O}(|G|)$. It is a plausible assumption that most specific graph representations can be transformed into our list-based representation without much effort. This ensures a certain generality of our upper complexity bounds in the sense that the corresponding algorithms are, to a large extent, independent from implementational details. Note also that the list-based structure only requires space linear in $|G|$.

In the adjacency list representation, we do not have random access to specific nodes in the graph database, or to specific neighbours of a given node. Thus, we have to measure a non-constant running-time for performing such operations. However, the algorithms for our upper bounds are independent from this aspect, i. e., the total running times would not change if we assume random access to nodes in constant time. Therefore, it is a stronger statement that a conditional lower bound is matched by an algorithm that *does not* exploit specific implementational aspects or advanced data structures, but works for basic graph representations.

An exception to this is Theorem 20, for which we can obtain some small improvement by applying the technique of lazy array initialization (see Remark 21).

2.2 Graph Databases and Regular Path Queries

A *nondeterministic finite automaton* (NFA for short) is a tuple $M = (G, S, T)$, where $G = (V, E)$ is a Σ-graph (the nodes $q \in V$ are also called *states*), $S \subseteq V$ with $S \neq \emptyset$ is the set of *start states* and $T \subseteq V$ with $T \neq \emptyset$ is the set of *final states*. The language $\mathcal{L}(M)$ of an NFA M is the set of all labels of paths from some start state to some final state. For a Σ-graph $G = (V, E)$, any subsets $S, T \subseteq V$ with $S \neq \emptyset \neq T$ induce the NFA (G, S, T). If $S = \{s\}$ and $T = \{t\}$ are singletons, then we also write (G, s, t) instead of $(G, \{s\}, \{t\})$.

The set RE_Σ of *regular expressions* (*over* Σ) is recursively defined as follows: $a \in \mathsf{RE}_\Sigma$ for every $a \in \Sigma \cup \{\varepsilon\}$; $(\alpha \cdot \beta) \in \mathsf{RE}_\Sigma$, $(\alpha \vee \beta) \in \mathsf{RE}_\Sigma$, and $(\alpha)^+ \in \mathsf{RE}_\Sigma$, for every $\alpha, \beta \in \mathsf{RE}_\Sigma$. For any $\alpha \in \mathsf{RE}_\Sigma$, let $\mathcal{L}(\alpha)$ be the regular language described by the regular expression α defined as usual: for every $a \in \Sigma \cup \{\varepsilon\}$, $\mathcal{L}(a) = \{a\}$, and for every $\alpha, \beta \in \mathsf{RE}_\Sigma$, $\mathcal{L}(\alpha \cdot \beta) = \mathcal{L}(\alpha) \cdot \mathcal{L}(\beta)$, $\mathcal{L}(\alpha \vee \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$ and $\mathcal{L}(\alpha^+) = \mathcal{L}(\alpha)^+$. We also use α^* as short hand form for $\alpha^+ \vee \varepsilon$. By $|\alpha|$, we denote the length of α represented as a string.

▶ **Proposition 2.** *Every regular expression α can be transformed in time $\mathrm{O}(|\alpha|)$ into an equivalent* NFA *$M = (G, p_0, p_f)$ with $|G| = \mathrm{O}(|\alpha|)$.*

In the following, when we speak about an *automaton* (or an NFA) *for a regular expression* α, we always mean an NFA equivalent to α with the properties asserted by Proposition 2.

A Σ-graph without ε-arcs is also called a *graph database* (*over* Σ); in the following, we denote graph databases by $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$. Since $V_\mathcal{D}$ is represented as a list, any graph database implicitly represents a linear order on $V_\mathcal{D}$ (i. e., the order induced by the list that represents $V_\mathcal{D}$), which we denote by $\preceq_\mathcal{D}$, or simply $\preceq$ if $\mathcal{D}$ is clear from the context. A graph database $\mathcal{D}$ is *sparse* if $|E_\mathcal{D}| = \mathrm{O}(|V_\mathcal{D}|)$.

Regular expressions q (over alphabet Σ) are interpreted as *regular path queries* (RPQ) for graph databases (over Σ). The *result* $q(\mathcal{D})$ of an RPQ q on a graph database $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$ over Σ is the set $q(\mathcal{D}) = \{(u, v) \mid u, v \in V_\mathcal{D}, \mathcal{L}((\mathcal{D}, u, v)) \cap \mathcal{L}(q) \neq \emptyset\}$.

If we interpret q as a *Boolean* RPQ, then the result is $q_{\mathsf{B}}(\mathcal{D}) = \mathsf{true}$ if $q(\mathcal{D}) \neq \emptyset$ and $q_{\mathsf{B}}(\mathcal{D}) = \mathsf{false}$ otherwise. We consider the following RPQ evaluation problems ($\mathcal{D}$ is a graph database, q an RPQ and u, v two nodes from $\mathcal{D}$):

Name	Input	Task
RPQ-Boole	$\mathcal{D}$, q	Decide whether $q_{\mathsf{B}}(\mathcal{D}) = \mathsf{true}$.
RPQ-Test	$\mathcal{D}$, q, u, v	Decide whether $(u, v) \in q(\mathcal{D})$.
RPQ-Witness	$\mathcal{D}$, q	Compute a witness $(u, v) \in q(\mathcal{D})$ or report that none exists.
RPQ-Eval	$\mathcal{D}$, q	Compute the whole set $q(\mathcal{D})$
RPQ-Count	$\mathcal{D}$, q	Compute $\lvert q(\mathcal{D})\rvert$.
RPQ-Enum	$\mathcal{D}$, q	Enumerate the whole set $q(\mathcal{D})$.

By *sorted RPQ-Enum* (or *semi-sorted* RPQ-Enum), we denote the variant of RPQ-Enum, where the pairs of $q(\mathcal{D})$ are to be enumerated in lexicographical order with respect to $\preceq_{\mathcal{D}}$ (or ordered only with respect to their left elements, while successive pairs with the same right element can be ordered arbitrarily, respectively).

▶ Remark 3. If an order $\preceq'$ on $V_{\mathcal{D}}$ is explicitly given as a bijection $\pi : V_{\mathcal{D}} \to \{1, \ldots, n\}$, then we can modify $\mathcal{D}$ (in $\mathrm{O}(|V_{\mathcal{D}}|)$) such that $\preceq_{\mathcal{D}} = \preceq'$. In this regard, sorted RPQ-Enum just models the case where we wish the enumeration to be sorted according to some order. In particular, by assuming the order $\preceq_{\mathcal{D}}$ to be implicitly represented by $\mathcal{D}$, we *do not* hide the non-linear complexity of sorting n numbers (this would only be the case if we would generally assume that $V_{\mathcal{D}} \subseteq \mathbb{N}$ and then require sorted enumeration with respect to $\leq$).

2.3 General Algorithmic Framework for RPQ Evaluation

We assume the RAM model with logarithmic word-size as our computational model. Let us next discuss our algorithmic framework for RPQ evaluation. The input to our algorithms is a graph database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ and an RPQ q (and, for solving the problem RPQ-Test, also a pair $(u, v) \in V_{\mathcal{D}}$).

In the case of RPQ-Enum, the algorithms have routines preprocess and enum. Initially, preprocess performs some preliminary computations on the input or constructs some auxiliary data structures; the performance of preprocess is measured in its running time depending on the input size as usual (i. e., we treat preprocess as an individual algorithm). Then enum will produce an enumeration $(u_1, v_1), (u_2, v_2), \ldots, (u_\ell, v_\ell)$ such that $q(\mathcal{D}) = \{(u_i, v_i) \mid 1 \leq i \leq \ell\}$, no element occurs twice, and the algorithm reports when the enumeration is done. We measure the performance of enum in terms of its *delay*, which describes the time that (in the worst-case) elapses between enumerating two consecutive elements, between the start of the enumeration and the first element, and between the last element and the end of the enumeration (or between start and end in case that $q(\mathcal{D}) = \emptyset$). We say that (variants of) RPQ-Enum can be solved *with preprocessing p and delay d*, where p and d are functions bounding the preprocessing running time and the delay. In the case that $p = \mathrm{O}(d)$, the preprocessing complexity is absorbed by the delay; in this case, we say that (variants of) RPQ-Enum can be solved *with delay d* and do not mention any bound on the preprocessing.

We also consider RPQ-Enum in the *dynamic setting*, i. e., there is the possibility to perform *update* operations on the input graph database $\mathcal{D}$, which triggers a routine update. After an update and termination of the update routine, invoking enum is supposed to enumerate $q(\mathcal{D}')$, where $\mathcal{D}'$ is the updated graph database. The performance of an algorithm for RPQ-Enum is then measured in the running times of routines preprocess (to be initially carried out only once) and update, as well as the delay. We only consider the following types

of individual updates: inserting a new arc between existing nodes, deleting an arc, adding a new (isolated) node, deleting an (isolated) node. In particular, deleting or adding a single non-isolated node u may require a non-constant number of updates.

3 The Product Graph Approach

The PG-approach has already been informally described in the introduction; for our fine-grained perspective, we need to define it in detail. Let $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$ be a graph database over some alphabet Σ and let q be an RPQ over Σ. Furthermore, let (G_q, p_0, p_f) with $G_q = (V_q, E_q)$ be an automaton for q. Recall that, according to Proposition 2, G_q can be obtained in time $O(|q|)$ and it has $O(|q|)$ states and $O(|q|)$ arcs. The *product graph* of $\mathcal{D}$ and G_q is the Σ-graph $G_\boxtimes(\mathcal{D}, q) = (V_\boxtimes(\mathcal{D}, q), E_\boxtimes(\mathcal{D}, q))$, where $V_\boxtimes(\mathcal{D}, q) = \{(u, p) \mid u \in V_\mathcal{D}, p \in V_q\}$ and

$$E_\boxtimes(\mathcal{D}, q) = \{((u, p), x, (v, p')) \mid (u, x, v) \in E_\mathcal{D}, (p, x, p') \in E_q\} \cup \{((u, p), \varepsilon, (u, p')) \mid u \in V_\mathcal{D}, (p, \varepsilon, p') \in E_q\}.$$

▶ Remark 4. The arc labels in $G_\boxtimes(\mathcal{D}, q)$ are superfluous in the sense that we only need the underlying graph of $G_\boxtimes(\mathcal{D}, q)$ (see Lemma 6). We define it nevertheless as Σ-graph, since then all our definitions and terminology for Σ-graphs introduced above apply as well.

▶ **Lemma 5.** *$|V_\boxtimes(\mathcal{D}, q)| = O(|V_\mathcal{D}||q|)$, $|E_\boxtimes(\mathcal{D}, q)| = O(|\mathcal{D}||q|)$ and $G_\boxtimes(\mathcal{D}, q)$ can be computed in time $O(|G_\boxtimes(\mathcal{D}, q)|) = O(|\mathcal{D}||q|)$.*

The following lemma, which is an immediate consequence of the construction, shows how $G_\boxtimes(\mathcal{D}, q)$ can be used for solving RPQ evaluation tasks (recall that E^* is the reflexive-transitive closure of the underlying unlabelled graph).

▶ **Lemma 6.** *For every $u, v \in V_\mathcal{D}$, $(u, v) \in q(\mathcal{D})$ if and only if $((u, p_0), (v, p_f)) \in (E_\boxtimes(\mathcal{D}, q))^*$.*

4 Fine-grained Complexity and Conditional Lower Bounds

We now state several computational problems along with hypotheses regarding their complexity, which are commonly used in the framework of fine-grained complexity to obtain conditional lower bounds. We discuss some details and give background information later on.

- Orthogonal Vectors (OV): Given sets A, B each containing n Boolean-vectors of dimension d, check whether there are vectors $\vec{a} \in A$ and $\vec{b} \in B$ that are orthogonal.
  OV-Hypothesis: For every $\epsilon > 0$ there is no algorithm solving OV in time $O(n^{2-\epsilon}\,\mathsf{poly}(d))$.
- Boolean Matrix Multiplication (BMM): Given Boolean $n \times n$ matrices A, B, compute $A \times B$.
  com-BMM-Hypothesis: For every $\epsilon > 0$ there is no combinatorial algorithm that solves BMM in time $O(n^{3-\epsilon})$.
- Sparse Boolean Matrix Multiplication (SBMM): Like BMM, but all matrices are represented as sets $\{(i, j) \mid A[i, j] = 1\}$ of 1-entries.
  SBMM-Hypothesis: There is no algorithm that solves SBMM in time $O(m)$ (where m is the total number of 1-entries, i. e., $m = |\{(i, j) \mid A[i, j] = 1\}| + |\{(i, j) \mid B[i, j] = 1\}| + |\{(i, j) \mid (A \times B)[i, j] = 1\}|$).
- Online Matrix-Vector Multiplication (OMv): Given Boolean $n \times n$-matrix M and a sequence $\vec{v}^1, \vec{v}^2, \ldots, \vec{v}^n$ of n-dimensional Boolean vectors, compute sequence $M\vec{v}^1, M\vec{v}^2, \ldots, M\vec{v}^n$, where $M\vec{v}^i$ is produced as output before $\vec{v}^{i+1}$ is received as input.
  OMv-Hypothesis: For every $\epsilon > 0$ there is no algorithm that solves OMv in time $O(n^{3-\epsilon})$.

We will reduce these problems to variants of RPQ evaluation problems in such a way that algorithms with certain running times for RPQ evaluation would break the corresponding hypotheses mentioned above. Thus, we obtain lower bounds for RPQ enumeration that are conditional to these hypotheses. In the following, we give a very brief overview of the relevance of these problems and corresponding hypotheses in fine-grained complexity.

The problem OV can be solved by brute-force in time $O(n^2d)$ and the hypothesis that there is no subquadratic algorithm is well-established. It exists in slightly different variants and has been formulated in several different places in the literature (e. g., [21, 22, 50]). The variant used here is sometimes referred to as *moderate dimension* OV-hypothesis in contrast to *low dimension* variants, where d can be assumed to be rather small in comparison to n. The relevance of the OV-hypothesis is due to the fact that it is implied by the Strong Exponential Time Hypothesis (SETH) [46, 47], and therefore it is a convenient tool to prove SETH lower bounds that has been applied in various contexts.

One of the most famous computational problems is BMM, which, unfortunately, is a much less suitable basis for conditional lower bounds. The straightforward algorithm solves it in time $O(n^3)$, but there are *fast matrix multiplication* algorithms that run in time $O(n^{2.373})$ [49, 30]. It is unclear how much further this exponent can be decreased and there is even belief that BMM can be solved in time $n^{2+o(1)}$ (see [48] and Section 6 of [13]). However, these theoretically fast algorithms cannot be considered efficient in a practical sense, which motivates the mathematically informal notion of "*combinatorial*" algorithms (see, e. g., [52]).[1] So far, no truly subcubic *combinatorial* BMM-algorithm exists and it has been shown in [52] that BMM is contained in a class of problems (including other prominent examples like Triangle Finding (also mentioned below) and Context-Free Grammar Parsing) which are all equivalent in the sense that if one such problem is solvable in truly subcubic time by a combinatorial algorithm, then all of them are. Consequently, it is often possible to argue that the existence of a certain combinatorial algorithm for some problem would imply a major (and unlikely) algorithmic breakthrough with respect to BMM, Parsing, Triangle Finding, etc. Despite the defect of relying on the vague notion of *combinatorial* algorithms, this lower bound technique is a common approach in fine-grained complexity (see, e. g., [52, 32, 2, 3, 1, 33]). Whenever we use the com-BMM-hypothesis, our reductions will always be combinatorial, which is necessary; moreover, whenever we say that a certain running time cannot be achieved unless the com-BMM-hypothesis fails, we mean, of course, that it cannot be achieved by a combinatorial algorithm.

In order to make BMM suitable as base problem for conditional lower bounds (that does not rely on *combinatorial* algorithms) one can formulate the weaker (i. e., more plausible) hypothesis that BMM cannot be solved in time linear in the number of 1-entries of the matrices (therefore called *sparse* BMM since matrices are represented in a sparse way); see [4, 54]. Another approach is to require the output matrix $A \times B$ to be computed column by column, i. e., formulating it as the online-version OMv. For OMv, subcubic algorithms are not known and would yield several major algorithmic breakthroughs (see [32]).

A convenient tool to deal with BMM is the problem Triangle: check whether a given undirected graph G has a triangle. This is due to the fact that these two problems are subcubic equivalent with respect to combinatorial algorithms (see [52]), i. e., the com-BMM-hypothesis fails if and only if Triangle can be solved by a combinatorial algorithm in time $O(n^{3-\epsilon})$ for some $\epsilon > 0$. Thus, for lower bounds conditional to the com-BMM-hypothesis, we

[1] The term "combinatorial algorithm" is not well-defined, but intuitively such algorithms have running times with low constants in the O-notation, and are feasibly implementable.

can make use of both these problems. There is also a (non-combinatorial) Triangle-hypothesis that states that Triangle cannot be solved in linear time in the number of edges, but we were not able to apply it in the context of RPQ evaluation (see [2] for different variants of Triangle).

5 Bounds for the Non-Enumeration Problem Variants

We now investigate how well the PG-approach performs with respect to the non-enumeration variants of RPQ evaluation, and we give some evidence that, in most cases, it can be considered optimal or almost optimal (subject to the algorithmic hypotheses of Section 4).

5.1 Boolean Evaluation, Testing and Computing a Witness

It is relatively straightforward to see that the problems RPQ-Test and RPQ-Boole are equivalent and can both be reduced to RPQ-Witness. Hence, upper bounds for RPQ-Witness and lower bounds for RPQ-Test or RPQ-Boole automatically apply to all three problem variants, which simplifies the proofs for such bounds.

The PG-approach directly yields the following upper bound.

▶ **Theorem 7.** *RPQ-Test, RPQ-Boole and RPQ-Witness can be solved in time* $O(|\mathcal{D}||q|)$.

More interestingly, we can complement this upper bound with lower bounds as follows.

▶ **Theorem 8.** *If any of the problems RPQ-Test, RPQ-Boole and RPQ-Witness can be solved in time* $O(|\mathcal{D}|^{2-\epsilon} + |q|^2)$ *or* $O(|\mathcal{D}|^2 + |q|^{2-\epsilon})$ *for some* $\epsilon > 0$*, then the OV-hypothesis fails. This lower bound also holds for the restriction to sparse graph databases.*

Proof Sketch. The overall reduction is similar to the reduction from [6] used for proving conditional lower bounds of regular expression matching: For an OV-instance $A = \{\vec{a}^1, \ldots, \vec{a}^n\}$ and $B = \{\vec{b}^1, \ldots, \vec{b}^n\}$, we define $q = \#(w_1 \vee w_2 \vee \ldots \vee w_n)\#$, where w_i is a bit-string representing vector $\vec{b}^i$, and we create a graph database that, for every $\vec{a}^i$, has a path whose j^{th} arc is labelled with 0 if $\vec{a}^i[j] = 1$ and with both 0 and 1 if $\vec{a}^i[j] = 0$. This yields a reduction that rules out running times of the form $O((|\mathcal{D}||q|)^{1-\epsilon})$; the stronger statement of the theorem can be obtained by carefully partitioning A or B depending on ϵ into several instances and then using the above reduction (with a more careful running time analysis). ◀

Since $(|\mathcal{D}||q|)^{1-\epsilon} \leq ((\max\{|\mathcal{D}|, |q|\})^2)^{1-\epsilon} = \max\{|\mathcal{D}|^{2-2\epsilon}, |q|^{2-2\epsilon}\} \leq |\mathcal{D}|^{2-\epsilon} + |q|^2$, Theorem 8 also rules out running times of the form $O((|\mathcal{D}||q|)^{1-\epsilon})$ and $O(\max\{|\mathcal{D}|, |q|\}^{2-\epsilon})$, but does not exclude a running time of $O(|\mathcal{D}|^{2-\epsilon} + f(|q|))$ with $f(|q|) = \Omega(|q|^2)$. However, for $\epsilon < 1$ this is super-linear in $|\mathcal{D}|$ (and therefore inferior to $O(|\mathcal{D}||q|)$ under the assumption $|q| \ll |\mathcal{D}|$), and for $\epsilon = 1$, we would obtain $O(|\mathcal{D}| + f(|q|))$, which, under the assumption $|q| \ll |\mathcal{D}|$, is a small and arguably negligible asymptotic improvement over $O(|\mathcal{D}||q|)$.

If the size of $\mathcal{D}$ is expressed in terms of $|V_\mathcal{D}|$, then Theorem 7 also gives an upper bound of $O(|V_\mathcal{D}|^2|q|)$. In this regard, we can show the following lower bound.

▶ **Theorem 9.** *If any of the problems RPQ-Test, RPQ-Boole and RPQ-Witness can be solved in time* $O(|V_\mathcal{D}|^{3-\epsilon} + |q|^{3-\epsilon})$ *for some* $\epsilon > 0$*, then the com-BMM-hypothesis fails.*

Proof Sketch. We reduce from Triangle (see Section 4). For a given graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, we create a graph database $\mathcal{D}$ by using 4 copies $V_i = \{(u, i) \mid u \in V\}$, $i \in [4]$, of V and add all of G's arcs between each V_i and V_{i+1}, $i \in [3]$, i. e., there is an arc

$((u,i),\mathsf{a},(v,i+1))$ if and only if $(u,v) \in E$ (a is a symbol). Now G contains a triangle if and only if $\mathcal{D}$ has a length-3 path from $(u,1)$ to $(v,4)$ with $u = v$. The problem is how to query paths that are synchronised in this way. To this end, we add a path $s_1, s_2, \ldots, s_n$ (each arc labelled with a) and, for every $j \in [n]$, an a-labelled arc from s_j to $(v_j, 1) \in V_1$; analogously, we add a path $t_1, t_2, \ldots, t_n$ with an arc from $(v_j, 4)$ to t_j. Now there is an a^{n+4}-labelled path from s_1 to t_n if and only if there is a length-3 path from $(u,1)$ to $(u,4)$ *for some* $u \in V$. ◀

Since $\mathrm{O}((|\mathcal{D}||q|)^{1-\epsilon}) \subseteq \mathrm{O}((|V_\mathcal{D}|^2|q|)^{1-\epsilon}) \subseteq \mathrm{O}(|V_\mathcal{D}|^{3-\epsilon} + |q|^{3-\epsilon})$, such running times are also ruled out under the com-BMM-hypothesis. Especially, a combinatorial algorithm with running time $\mathrm{O}((|\mathcal{D}||q|)^{1-\epsilon})$ refutes *both* the OV- and the com-BMM-hypothesis; thus, such an algorithm does not exist provided that at least one of these hypotheses is true (basing lower bounds on several hypotheses is common in fine-grained complexity, see, e. g., [3]).

The lower bounds discussed above are only meaningful for combined complexity. However, the upper bound of Theorem 7 already yields the optimum of *linear* data complexity.

5.2 Full Evaluation and Counting

The following upper bound is again a straightforward application of the PG-approach.

▶ **Theorem 10.** *RPQ-Eval can be solved in time* $\mathrm{O}(|V_\mathcal{D}||\mathcal{D}||q|)$.

Instead of using graph-searching techniques on $G_\boxtimes(\mathcal{D}, q)$, we could also compute the complete transitive closure of $G_\boxtimes(\mathcal{D}, q)$ with fast matrix multiplication.

▶ **Theorem 11.** *If BMM can be solved in time* $\mathrm{O}(n^\omega)$ *with* $\omega \geq 2$*, then RPQ-Eval can be solved in time* $\mathrm{O}(|V_\mathcal{D}|^\omega |q|^\omega)$.

We mention this theoretical upper bound for completeness, but stress the fact that our main interest lies in *combinatorial* algorithms.

In addition to the practical limitations of fast matrix multiplication, we also observe that the approach of Theorem 11 is only better if the graph database is not too sparse, i. e., only if $|V_\mathcal{D}||\mathcal{D}| = \Omega(|V_\mathcal{D}|^\omega)$.

Next, we investigate the question whether $\mathrm{O}(|V_\mathcal{D}||\mathcal{D}||q|)$ is optimal for RPQ-Eval at least with respect to combinatorial algorithms. Since for RPQ-Eval the PG-approach does not yield an algorithm that is linear in data complexity (like it was the case with respect to the problems of Section 5.1), the question arises whether the $|V_\mathcal{D}||\mathcal{D}|$ part can be improved at the cost of spending more time in $|q|$. It seems necessary that respective *data complexity lower bounds* need reductions that do not use q to represent a non-constant part of the instance, as it was the case for both the OV and the Triangle reduction from Section 5.1.

It is not difficult to see that the product of two $n \times n$ Boolean matrices A and B can be represented by node sets $V_i = \{(j,i) \mid j \in [n]\}$, $i \in [3]$, with arcs $((j,1),\mathsf{a},(k,2))$ iff $A[j,k] = 1$, and $((k,2),\mathsf{a},(\ell,3))$ iff $B[k,\ell] = 1$, and the RPQ $q = \mathsf{aa}$, i. e., $((i,1),(j,3)) \in q(\mathcal{D})$ iff $(A \times B)[i,j] = 1$ (it is, of course, crucial that q has constant size). This shows that, for combinatorial algorithms and subject to the com-BMM-hypothesis, $\mathrm{O}(|V_\mathcal{D}||\mathcal{D}|)$ is a tight bound for the data complexity of RPQ-Eval.

▶ **Theorem 12.** *If RPQ-Eval can be solved in time* $\mathrm{O}((|V_\mathcal{D}||\mathcal{D}|)^{1-\epsilon} f(|q|))$ *for some function* f *and some* $\epsilon > 0$*, then the com-BMM-hypothesis fails.*

If we drop the restriction to combinatorial algorithms, we can nevertheless show (with more or less the same construction) that linear time in data complexity is impossible, unless the SBMM-hypothesis fails. However, since the size of the output $q(\mathcal{D})$ might be super-linear in $|\mathcal{D}|$, we should interpret *linear* as linear in $|\mathcal{D}| + |q(\mathcal{D})|$.

▶ **Theorem 13.** *If RPQ-Eval can be solved in time* $O((|q(\mathcal{D})| + |\mathcal{D}|)f(|q|))$ *for some function* f*, then the SBMM-hypothesis fails.*

Surprisingly, we can obtain a more complete picture for the problem RPQ-Count. First, we observe that obviously all upper bounds carry over from RPQ-Eval to RPQ-Count. On the other hand, a combinatorial $O((|V_{\mathcal{D}}||\mathcal{D}|)^{1-\epsilon}f(q))$ algorithm or a general $O((|q(\mathcal{D})| + |\mathcal{D}|)f(|q|))$ algorithm for RPQ-Count does not seem to help for solving Boolean matrix multiplication (and therefore, the lower bounds do not carry over). Fortunately, it turns out that OV is a suitable problem to reduce to RPQ-Count, although by a rather different reduction compared to the one used for Theorem 8.

▶ **Theorem 14.** *If RPQ-Count can be solved in time* $O(|\mathcal{D}|^{2-\epsilon}f(|q|))$ *for some function* f *and* $\epsilon > 0$*, then the OV-hypothesis fails.*

Proof Sketch. Let $A = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n\}$ and $B = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n\}$ be an OV-instance, and let A' be the Boolean matrix having rows $\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n$ and let B' be the Boolean matrix having columns $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n$. Then $(A' \times B')[i, j] = 0$ if and only if $\vec{a}_i$ and $\vec{b}_j$ are orthogonal. If we represent $A' \times B'$ as a graph database and a size-2 RPQ q (as sketched above), we have that $|q(\mathcal{D})|$ is the number of 1-entries in $A' \times B'$ and therefore we can solve the OV-instance (A, B) by solving RPQ-Count for $\mathcal{D}$ and q. ◀

Since Theorem 14 also excludes running time $O((|V_{\mathcal{D}}||\mathcal{D}|)^{1-\epsilon}f(|q|))$ for any function f and $\epsilon > 0$ (without restriction to combinatorial algorithms), it also shows that, subject to the OV-hypothesis, $O(|V_{\mathcal{D}}||\mathcal{D}|)$ is a tight bound for the data complexity of RPQ-Count.

6 Bounds for the Enumeration of RPQs

By using the PG-approach for enumeration, we can obtain the following upper bound.

▶ **Theorem 15.** *Sorted RPQ-Enum can be solved with* $O(|\mathcal{D}||q|)$ *delay and* $O(1)$ *updates.*

Proof Sketch. We store all nodes (u, p_0) of $G_{\boxtimes}(\mathcal{D}, q)$ in an array S, all nodes (v, p_f) in an array T, and the general idea is to perform an individual BFS from each node in S. However, in order to bound the delay, we first have to determine the subset $S' \subseteq S$ of nodes that can reach at least one node from T. To this end, we add a new node v_{sink} with incoming arcs from each node in T, and then we perform a BFS from v_{sink} in reverse direction. This concludes the preprocessing (and, with a bit of care, this can be done in time $O(|G_{\boxtimes}(\mathcal{D}, q)|) = O(|\mathcal{D}||q|)$). In the enumeration, we perform a BFS from each $(u, p_0) \in S'$ and we consider these start nodes ordered by $\preceq$. In each individual BFS, we first collect in an array all visited nodes $(v_1, p_f), (v_2, p_f), \ldots$ and then enumerate all pairs (u, v_i) sorted by their right element according to $\preceq$. Due to our preprocessing, we produce at least one pair in each BFS, and each BFS can be done in time $O(|G_{\boxtimes}(\mathcal{D}, q)|) = O(|\mathcal{D}||q|)$, including the time needed for enumerating the pairs. Updates can be maintained trivially, since the delay bound allows to repeat the preprocessing with respect to the updated database. ◀

This enumeration algorithm is easy to implement and has some nice features like linear preprocessing (in data complexity), sorted enumeration and constant updates. Unfortunately, these features come more or less for free with the disappointing delay bound. Is the PG-approach therefore the wrong tool for RPQ-enumeration? Or can we give evidence that linear delay is a barrier we cannot break? The rest of this work is devoted to this question.

Since running an algorithm for RPQ-Enum until we get the first element yields an algorithm for RPQ-Boole (with preprocessing plus delay as running time), and since running such an algorithm completely solves RPQ-Eval in time preprocessing plus $|q(\mathcal{D})|$ times delay, we can inherit several lower bounds directly from Section 5.

▶ **Theorem 16.** *If, for some function f and $\epsilon \geq 0$, RPQ-Enum can be solved with*

1. *delay $O(|\mathcal{D}|^{2-\epsilon} + |q|^2)$ or $O(|\mathcal{D}|^2 + |q|^{2-\epsilon})$, then the OV-conjecture fails.*
2. *delay $O(|V_\mathcal{D}|^{3-\epsilon} + |q|^{3-\epsilon})$, then the com-BMM-hypothesis fails.*
3. *preprocessing $O(|\mathcal{D}|f(|q|))$ and delay $O(f(|q|))$, then the SBMM-hypothesis fails.*
4. *prep. $O(|V_\mathcal{D}|^{3-\epsilon}f(|q|))$ and delay $O(|V_\mathcal{D}|^{1-\epsilon}f(|q|))$, then the com-BMM-hypothesis fails.*

The first two bounds only tell us that we might not be able to lower the delay $O(|\mathcal{D}||q|)$ in terms of combined complexity. While this point of view was justified for the problems discussed in Section 5, it does not say anything regarding delays of the form $O(|\mathcal{D}|^{1-\epsilon}f(|q|))$. The third bound, conditional to the SBMM-hypothesis, is much more relevant, since it suggests that the optimum of linear preprocessing and constant delay is not reachable. For combinatorial algorithms, the fourth bound at least answers our main question with $|\mathcal{D}|$ replaced by $|V_\mathcal{D}|$: with linear preprocessing, we cannot get below a delay of $O(|V_\mathcal{D}|)$.

These lower bounds can be improved significantly, if we also want to handle updates (within some reasonable time bounds).

▶ **Theorem 17.** *If RPQ-Enum can be solved with*

1. *arbitrary preprocessing, $O(|V_\mathcal{D}|^{1-\epsilon}f(|q|))$ updates and $O(|V_\mathcal{D}|^{1-\epsilon}f(|q|))$ delay for some function f and $\epsilon \geq 0$, then the OMv-hypothesis fails.*
2. *$O(|V_\mathcal{D}|^{3-\epsilon}f(|q|))$ preprocessing, $O(|V_\mathcal{D}|^{2-\epsilon}f(|q|))$ updates and $O(|V_\mathcal{D}|^{2-\epsilon}f(|q|))$ delay for some function f and $\epsilon \geq 0$, then the com-BMM-hypothesis fails.*

Proof Sketch. OMv-reduction: Let M, $\vec{v}^1, \vec{v}^2, \ldots, \vec{v}^n$ be an OMv instance. We set $q = \mathtt{aa}$ and start with an empty graph database, which, by $O(n^2)$ updates, is turned into $\mathcal{D}_{M,\vec{v}^1} = (\{u_i, v_i, w \mid 1 \leq i \leq n\}, \{(u_i, \mathtt{a}, v_j) \mid M[i,j] = 1\} \cup \{(v_j, \mathtt{a}, w) \mid \vec{v}^1[j] = 1\})$. Enumerating $q(\mathcal{D}_{M,\vec{v}^1})$ gives $\{(u_i, w) \mid (M\vec{v}^1)[i] = 1\}$ which represent $M\vec{v}^1$. Analogously, we produce $M\vec{v}^i$ with $q(\mathcal{D}_{M,\vec{v}^i})$ for each $2 \leq i \leq n$, where $\mathcal{D}_{M,\vec{v}^i}$ can be obtained from $\mathcal{D}_{M,\vec{v}^{i-1}}$ by n updates.

Triangle-reduction: The idea is similar to the reduction of Theorem 9. We transform a graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ into a graph database $\mathcal{D}_{v_1}$ with 4 copies $V_i = \{(u, i) \mid u \in V\}$, $i \in [4]$, of V and an arc $((u,i), \mathtt{a}, (v, i+1))$ if and only if $(u,v) \in E$. Instead of appending length n-paths at V_1 and V_4 (as done for Theorem 9), we only add arcs $(s, \mathtt{a}, (v_1, 1))$, $((v_1, 4), \mathtt{a}, t)$ and set $q = \mathtt{aaaaa}$. Now $q(\mathcal{D}_{v_1}) = \{(s,t)\}$ if there is a triangle that contains v_1, and $q(\mathcal{D}_{v_1}) = \emptyset$ otherwise. For every $2 \leq i \leq n$, we can obtain $\mathcal{D}_{v_i}$ from $\mathcal{D}_{v_{i-1}}$ by performing a constant number of updates, and then enumerate $q(\mathcal{D}_{v_i})$. ◀

The first bound rules out that we can get below $O(|V_\mathcal{D}|)$ for delay *and* update time, regardless of the preprocessing; the second one analogously rules out anything below $O(|\mathcal{D}|)$ (for combinatorial algorithms and linear preprocessing). While all these lower bounds suggest that improving on the linear delay may be rather difficult, they leave the following case open.

▶ **Question 18.** *Can RPQ-Enum be solved with $O(|\mathcal{D}|)$ preprocessing and $O(|V_\mathcal{D}|)$ delay in data complexity?*

What is a reasonable conjecture with respect to this question? A combinatorial algorithm that answers it in the positive does not seem to have any unlikely consequences. Indeed, it would just entail an $O(|\mathcal{D}||V_\mathcal{D}|)$ algorithm for RPQ-Eval (which exactly fits to Theorem 10), an $O(|V||G|)$ algorithm for computing transitive closures and an $O(n^3)$ algorithm

for multiplying Boolean $n \times n$ matrices, which is the state of the art for combinatorial algorithms (this is due to the obvious reductions from these problems to RPQ-Enum). However, since Question 18 is about enumeration, a positive answer also means that after $O(|G|)$ preprocessing, we can enumerate the transitive closure of a graph with delay $O(|V|)$, and that after $O(n^2)$ preprocessing, all 1-entries of the Boolean matrix multiplication can be enumerated with delay $O(n)$. Are such enumeration algorithms unlikely, so that we should rather expect a negative answer to Question 18? In fact not, since for the simple RPQs $q = \mathsf{a}^*$ or $q = \mathsf{aa}$, which are sufficient to encode transitive closures and Boolean matrix multiplications (see also the sketch on page 11), linear preprocessing and delay $O(|V_\mathcal{D}|)$ is indeed possible, as we shall obtain as byproducts of the results in the next section.

We close this section by the following remark that points out some similarity (and differences) of reductions used in Sections 5 and 6.

▶ Remark 19. As already mentioned in Section 5.1, the reduction used for the combined complexity lower bounds of Theorems 8 (and therefore Point 1 of Theorem 16) is similar to the reduction from [6] used for proving conditional lower bounds of regular expression matching. Moreover, the OV-lower bound for RPQ-Count of Theorem 14 is similiar to a lower bound on counting the results of certain conjunctive queries from [14], and the OMv-lower bound from Point 1 of Theorem 17 is similiar to a lower bound on enumerating certain conjunctive queries from [14].

The quite simple observation that Boolean matrix multiplication can be expressed as querying a bipartite graph (with conjunctive queries) has also been used in [8] (see also [13, Section 6]) and is also the base for the OMv-lower bound of [14]. In the context of this paper, this connection has been used in the bounds of Theorems 12, 13 and Points 3 and 4 of Theorem 16.

The obvious connection between evaluating (non-acyclic) conjunctive queries and finding triangles (or larger cliques) has already been observed in [20] (see also [13, Section 6]). However, the Triangle-lower bounds of this paper are quite different, since RPQs cannot explicitly express the structure of a triangle (or a larger clique, for that matter) by using conjunction. Therefore, our respective lower bounds (Theorems 9, Point 2 of Theorem 16, and Point 2 of Theorem 17) need to encode this aspect in a different way. With respect to Theorems 9 and Point 2 of Theorem 16 this is done by using non-constant queries (which explains why the lower bounds are not for data complexity, in contrast to the case of conjunctive queries), and with respect to Point 2 of Theorem 17, which *does* work for data complexity, it is done by using updates. Moreover, our Triangle-lower bounds do not seem to extend to larger cliques like it is the case for conjunctive queries (see [13, Section 6]).

Finally, we wish to point out that although some of the reductions used in this paper are similar to reductions used in the context of conjunctive queries, due to the difference of RPQs and CQs, none of the lower bounds directly carry over. Furthermore, note that the lower bound reductions in [8, 14] have been used for obtaining dichotomies and therefore have been stated in a much more general way.

7 Enumeration with Sub-Linear Delay

We now explore three different approaches towards enumeration of $q(\mathcal{D})$ with delay strictly better than $O(|\mathcal{D}|)$ (in data complexity): (1) allowing super-linear preprocessing, (2) enumerating a representative subset of $q(\mathcal{D})$, and (3) restricting the RPQs.

Regarding the first approach, we can improve the delay from $O(|\mathcal{D}|)$ to $O(|V_\mathcal{D}|)$ by increasing the linear preprocessing by a factor of $\overline{\Delta}(\mathcal{D}) \log(\overline{\Delta}(\mathcal{D}))$ (in data complexity). Recall that $\overline{\Delta}(\mathcal{D})$ is the average degree of $\mathcal{D}$.

▶ **Theorem 20.** *Sorted RPQ-Enum can be solved with* $\mathrm{O}(|q|^2 \log(\overline{\Delta}(\mathcal{D})|q|)\, \overline{\Delta}(\mathcal{D})|\,\mathcal{D}\,|)$ *preprocessing and* $\mathrm{O}(|V_\mathcal{D}|)$ *delay.*

Proof Sketch. The basic idea is to compute in the preprocessing for every $u \in V_\mathcal{D}$ a set A_u of all or at most $\overline{\Delta}(\mathcal{D})|q|$ nodes v such that $(u, v) \in q(\mathcal{D})$. With this information, we can then in the enumeration do the following for every $u \in V_\mathcal{D}$. If $|A_u| < \overline{\Delta}(\mathcal{D})|q|$, then A_u must contain *all* nodes reachable by a q-path from u, which we can therefore enumerate with constant delay. If $|A_u| = \overline{\Delta}(\mathcal{D})|q|$, then, by producing every $|V_\mathcal{D}|$ steps a pair (u, v) with $v \in A_u$, we can afford to run a whole BFS on the product graph without exceeding the delay of $|V_\mathcal{D}|$, which allows us to compute all remaining nodes reachable by a q-path from u.

In order to compute these sets A_u in the preprocessing, we first compute a DAG H of the strongly connected components of $G_\boxtimes(\mathcal{D}, q)$. Then we move through the connected components $1, 2, \ldots, \ell$ in a reverse topological sorting and compute for each component i a set B_i of at most $\overline{\Delta}(\mathcal{D})|q|$ nodes (v, p_f) (contained in earlier components $1, \ldots, i-1$) reachable from component i. This is done by initialising each B_i with the nodes (v, p_f) present in component i and then copying these sets along the backward arcs when moving through the reverse topological sorting. To avoid duplicates and to include only the smallest $\overline{\Delta}(\mathcal{D})|q|$ nodes in the sets B_i (the latter is required for sorted enumeration), we handle the sets by binary search trees with insert, delete and look-up operations in time $\mathrm{O}(\log(\overline{\Delta}(\mathcal{D})|q|))$. ◀

Obviously, in the worst case we can have $\overline{\Delta}(\mathcal{D}) = \Omega(|V_\mathcal{D}|)$ and then the preprocessing of the algorithm of Theorem 20 is $\Omega(|V_\mathcal{D}||\,\mathcal{D}\,|)$ (in data complexity) and therefore no improvement over just computing the complete set $q(\mathcal{D})$ in time $\mathrm{O}(|V_\mathcal{D}||\,\mathcal{D}\,|)$. However, for graph databases with low average degree, we can decrease the delay significantly from $\mathrm{O}(|\,\mathcal{D}\,|)$ to $\mathrm{O}(|V_\mathcal{D}|)$ at the cost of a slightly super-linear preprocessing time. As another remark about Theorem 20, we observe that the pre-computed information becomes worthless if $\mathcal{D}$ is updated. Finally, we mention a minor modification of the algorithm of Theorem 20 which yields a slightly different result that is interesting in its own right.

▶ Remark 21. At the cost of additional space (in $O(|V_\mathcal{D}|^2)$) and giving up on sorted enumeration, we can use the lazy array initialisation technique (see, e. g., the textbook [42]) in order to reduce the preprocessing time in Theorem 20 by a factor $\log(\overline{\Delta}(\mathcal{D})|q|)$ to $\mathrm{O}(|q|^2\, \overline{\Delta}(\mathcal{D})|\,\mathcal{D}\,|)$. More precisely, this can be achieved by implementing the sets A_u by arrays with lazy initialization instead of binary search trees.

Let us move on to the second approach. Evaluating an RPQ on a graph database $\mathcal{D}$ aims to find for each node u *all* its q-successors, i. e., nodes reachable by a q-path. It is therefore a natural restriction to ask for only *at least one* (if any) such successor. Likewise, we could also ask for at least one (if any) q-predecessor for every node. More precisely, instead of the whole set $q(\mathcal{D})$, the task is to enumerate a $q(\mathcal{D})$-*approximation*, which is a set $A \subseteq q(\mathcal{D})$ such that, for every $u, v \in V_\mathcal{D}$, if $(u, v) \in q(\mathcal{D})$, then also $(u, v'), (u', v) \in A$ for some $u', v' \in V_\mathcal{D}$. Such a set is representative for $q(\mathcal{D})$, since it contains for *every* node the information, whether it is involved as a source and whether it is involved as a target in some reachable pair from $q(\mathcal{D})$. The problem of enumerating any $q(\mathcal{D})$-approximation for given $\mathcal{D}$ and q will be denoted by App-RPQ-Enum.

▶ **Theorem 22.** *App-RPQ-Enum can be solved with linear preprocessing and constant delay.*

Proof Sketch. We store all nodes (u, p_0) of $G_\boxtimes(\mathcal{D}, q)$ in an array S and all nodes (v, p_f) in an array T, and we add a new node v_{source} with an arc to each $(u, p_0) \in S$. Then, we compute an array S', such that, for every $(v, p) \in V_\boxtimes(\mathcal{D}, q)$, if $S'[(v, p)] = 0$, then (v, p) is not reachable from any $(u, p_0) \in S$; if $S'[(v, p)] \neq 0$, then $S'[(v, p)]$ stores some $u \in V_\mathcal{D}$ such that

(v, p) is reachable from $(u, p_0) \in S$. To compute S' in time $O(|G_{\boxtimes}(\mathcal{D}, q)|)$ we first initialise $S'[(u, p_0)] = u$ for every $(u, p_0) \in S$, and then we perform a BFS from v_{source} and always set $S'[(u', p')] = S'[(u, p)]$ for every traversed arc $((u, p), (u', p'))$. By a BFS in the opposite direction from a new node v_{sink} connected to all $(v, p_f) \in T$, we can produce an analogous array T' (storing for each node from $V_{\boxtimes}(\mathcal{D}, q)$ a reachable node from T). With S' and T', we can now enumerate a $q(\mathcal{D})$-approximation with constant delay. ◀

Interestingly, it seems rather difficult to also support updates while keeping a low delay (the following bounds are due to the same reductions used for Theorem 17, simply because these reductions produce instances for which all $q(\mathcal{D})$-approximations are equal to $q(\mathcal{D})$).

▶ **Theorem 23.** *If* App-RPQ-Enum *can be solved with*

- *arbitrary preprocessing,* $O(|V_{\mathcal{D}}|^{1-\epsilon} f(|q|))$ *updates and* $O(|V_{\mathcal{D}}|^{1-\epsilon} f(|q|))$ *delay, then the OMv-hypothesis fails.*
- $O(|V_{\mathcal{D}}|^{3-\epsilon} f(|q|))$ *preprocessing,* $O(|V_{\mathcal{D}}|^{2-\epsilon} f(|q|))$ *updates and* $O(|V_{\mathcal{D}}|^{2-\epsilon} f(|q|))$ *delay for some function* f *and* $\epsilon \geq 0$*, then the com-BMM-hypothesis fails.*

Finally, as our third approach, we show that for restricted RPQ, we can solve RPQ-Enum with delay much smaller than $O(|\mathcal{D}|)$. We first need some definitions. For any class $Q \subseteq \mathsf{RPQ}$, we denote by Enum(Q) the problem RPQ-Enum where the input RPQ is from Q. Moreover, $\bigvee(Q)$ is the set of all RPQs of the form $(q_1 \vee \ldots \vee q_m)$ with $q_i \in Q$ for every $i \in [m]$. An RPQ q over Σ is a *basic transitive* RPQ (over Σ) if $q = (x_1 \vee \ldots \vee x_k)^*$ or $q = (x_1 \vee \ldots \vee x_k)^+$, where $x_1, \ldots, x_k \in \Sigma$; and q is a *short* RPQ (over Σ) if $q = (x_1 \vee \ldots \vee x_k)$ or $q = (x_1 \vee \ldots \vee x_k)(y_1 \vee \ldots \vee y_{k'})$, where $x_1, \ldots, x_k, y_1, \ldots, y_{k'} \in \Sigma$. By BT-RPQ and S-RPQ, we denote the class of basic transitive RPQ and the class of short RPQ, respectively.

We show that for the class $\bigvee(\mathsf{S\text{-}RPQ} \cup \mathsf{BT\text{-}RPQ})$ (which, e. g., contains RPQs of the form $q = (\mathtt{ab} \vee \mathtt{c}^* \vee \mathtt{b}(\mathtt{c} \vee \mathtt{d}) \vee (\mathtt{a} \vee \mathtt{b} \vee \mathtt{d})^+)$), semi-sorted RPQ-Enum can be solved with linear preprocessing and delay $O(\Delta(\mathcal{D}))$ in data complexity (recall *semi-sorted* from Section 2).

▶ **Theorem 24.** *Semi-sorted* Enum($\bigvee(\mathsf{S\text{-}RPQ} \cup \mathsf{BT\text{-}RPQ})$) *can be solved with preprocessing* $O(|q|^2|\mathcal{D}|)$ *and delay* $O(|q|^2\Delta(\mathcal{D}))$.

Proof Sketch. Semi-sorted Enum(BT-RPQ): This reduces to enumerating the transitive closure of a directed graph, which can be done by starting a BFS in each $u \in V_{\mathcal{D}}$ (ordered by $\preceq$): we only have to make sure that whenever we dequeue a node v, then we first visit (and possibly enqueue) its neighbours before producing (u, v). (See also [24].)

Semi-sorted Enum(S-RPQ): Let $q = (x_1 \vee \ldots \vee x_k)(y_1 \vee \ldots \vee y_{k'})$ (the case $q = (x_1 \vee \ldots \vee x_k)$ is trivial) and note that $V_{\boxtimes}(\mathcal{D}, q)$ is partitioned into node sets V_1, V_2 and V_3 (i. e., by the right elements of the nodes, since the NFA for q has three states) and we have to enumerate all reachable pairs (u, v) from $V_1 \times V_3$. In the preprocessing, we remove all nodes from V_2 with in- or out-degree 0, and then we remove all isolated nodes from V_1 and V_3. In the enumeration, we perform a BFS from each $u \in V_1$ (ordered by $\preceq$), but we store the reached nodes from V_3 in a queue and produce (and remove from the queue) a pair whenever $\Delta(\mathcal{D})$ steps are made by the BFS after the last output. It can be shown that the queue will not run dry prematurely and therefore the delay is bounded by $\Delta(\mathcal{D})$.

Semi-sorted Enum($\bigvee(\mathsf{S\text{-}RPQ} \cup \mathsf{BT\text{-}RPQ})$): We show a general meta-result: if for some $Q \subseteq \mathsf{RPQ}$ we can solve semi-sorted Enum(Q) with preprocessing p and delay d, then we can solve semi-sorted Enum($\bigvee(Q)$) with preprocessing $O(|q|p)$ and delay $O(|q|d)$. With the results from above, this directly implies the statement of the theorem. Let $q = (q_1 \vee \ldots \vee q_m) \in \bigvee(Q)$ with enumeration algorithms A_j for q_j. The idea is to initially perform the preprocessings for

all A_j and then run their enumerations in parallel in phases $1, 2, \ldots, |V_\mathcal{D}|$, where in phase i we get from each A_j exactly the pairs with left element i (for simplicity, assume $V_\mathcal{D} = [n]$ and $\preceq = \leq$). This is only possible since all A_j are semi-sorted. In phase i, we request the next elements from A_j, we discard those already produced as output and store all others as potential outputs. After we have done this, we output one of our potential outputs and then request the next elements until all A_j have finished their phase i. Since we cannot assume that we always have a new element among the m elements received next from the A_j, we have to show that we always have at least one potential output left when needed. ◀

Since $q = \mathtt{aa} \in$ S-RPQ is sufficient to express BMM as RPQ evaluation (see also page 11), Theorem 24 implies that enumerating (the 1-entries of) Boolean matrix products can be solved with linear preprocessing and $O(n)$ delay, but, on the other hand, this also immediately implies a matching data complexity lower bound for the upper bound of Theorem 24.

▶ **Theorem 25.** *If RPQ-Enum*(S-RPQ) *can be solved with prep.* $O(|V_\mathcal{D}|^{3-\epsilon} f(|q|))$ *and delay* $O(|\,\Delta(\mathcal{D})|^{1-\epsilon} f(|q|))$ *for some function* f *and* $\epsilon \geq 0$*, then the com-BMM-hypothesis fails.*

Compared to the full class of RPQs, the class $\bigvee$(S-RPQ $\cup$ BT-RPQ) is quite restricted. However, comprehensive experimental analyses of query logs suggest that quite restricted RPQs are still practically relevant: in the corpus of more than 50 million RPQs analysed in [18, Table 4], roughly 50% of the RPQs are from BT-RPQ and another 25% are of the form $q = x_1 x_2 \ldots x_k$, many of which with $k \leq 2$ making them S-RPQs [36].

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.
2. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014. doi:10.1109/FOCS.2014.53.
3. Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.
4. Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, pages 121–126, 2009. doi:10.1145/1514894.1514909.
5. Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017. doi:10.1145/3104031.
6. Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466, 2016. doi:10.1109/FOCS.2016.56.
7. Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 261–272, 2013.
8. Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, pages 208–222, 2007.

9 Jean-François Baget, Meghyn Bienvenu, Marie-Laure Mugnier, and Michaël Thomazo. Answering conjunctive regular path queries over guarded existential rules. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 793–799, 2017. doi:10.24963/ijcai.2017/110.

10 Pablo Barceló. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 175–188, 2013.

11 Pablo Barceló, Diego Figueira, and Miguel Romero. Boundedness of conjunctive regular path queries. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 104:1–104:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.104.

12 Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems (TODS)*, 37(4):31:1–31:46, 2012.

13 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636.

14 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 303–318, 2017.

15 Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. Regular path queries in lightweight description logics: Complexity and algorithms. *J. Artif. Intell. Res.*, 53:315–374, 2015. doi:10.1613/jair.4577.

16 Meghyn Bienvenu and Michaël Thomazo. On the complexity of evaluating regular path queries over linear existential rules. In *Web Reasoning and Rule Systems - 10th International Conference, RR 2016, Aberdeen, UK, September 9-11, 2016, Proceedings*, pages 1–17, 2016. doi:10.1007/978-3-319-45276-0_1.

17 Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *PVLDB*, 11(2):149–161, 2017.

18 Angela Bonifati, Wim Martens, and Thomas Timm. Navigating the maze of wikidata query logs. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 127–138, 2019. doi:10.1145/3308558.3313472.

19 Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *VLDB J.*, 29(2-3):655–679, 2020. doi:10.1007/s00778-019-00558-9.

20 Johann Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre. (The relevance of the list: propositional logic and complexity of the first order)*. PhD thesis, University of Caen Normandy, France, 2013. URL: https://tel.archives-ouvertes.fr/tel-01081392.

21 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

22 Karl Bringmann. Fine-grained complexity theory (tutorial). In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 4:1–4:7, 2019. doi:10.4230/LIPIcs.STACS.2019.4.

23 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

24 Katrin Casel, Tobias Friedrich, Stefan Neubert, and Markus L. Schmid. Shortest distances as enumeration problem. *CoRR*, abs/2005.06827, 2020. arXiv:2005.06827.

25 Katrin Casel and Markus L. Schmid. Fine-grained complexity of regular path queries. *CoRR*, abs/2101.01945, 2021. arXiv:2101.01945.

26 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987*, pages 323–330, 1987.

27 Diego Figueira. Containment of UC2RPQ: the hard and easy cases. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*, pages 9:1–9:18, 2020. doi:10.4230/LIPIcs.ICDT.2020.9.

28 Diego Figueira, Adwait Godbole, Shankara Narayanan Krishna, Wim Martens, Matthias Niewerth, and Tina Trautner. Containment of simple regular path queries. *CoRR*, abs/2003.04411, 2020. arXiv:2003.04411.

29 Dominik D. Freydenberger and Nicole Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. *J. Comput. Syst. Sci.*, 79(6):892–909, 2013. doi:10.1016/j.jcss.2013.01.008.

30 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

31 Grzegorz Gluch, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. The first order truth behind undecidability of regular path queries determinacy. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 15:1–15:18, 2019.

32 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30, 2015. doi:10.1145/2746539.2746609.

33 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 26:1–26:31, 2017. doi:10.4230/LIPIcs.ITCS.2017.26.

34 Leonid Libkin, Wim Martens, and Domagoj Vrgoc. Querying graphs with data. *Journal of the ACM*, 63(2):14:1–14:53, 2016.

35 Katja Losemann and Wim Martens. The complexity of regular expressions and property paths in SPARQL. *ACM Transactions on Database Systems (TODS)*, 38(4):24:1–24:39, 2013.

36 Wim Martens. Personal communication by email, October 28, 2019.

37 Wim Martens, Matthias Niewerth, and Tina Trautner. A trichotomy for regular trail queries. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, pages 7:1–7:16, 2020. doi:10.4230/LIPIcs.STACS.2020.7.

38 Wim Martens and Tina Trautner. Evaluation and enumeration problems for regular path queries. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, pages 19:1–19:21, 2018.

39 Wim Martens and Tina Trautner. Bridging theory and practice with query log analysis. *SIGMOD Rec.*, 48(1):6–13, 2019. doi:10.1145/3371316.3371319.

40 Wim Martens and Tina Trautner. Dichotomies for evaluating simple regular path queries. *ACM Trans. Database Syst.*, 44(4):16:1–16:46, 2019. doi:10.1145/3331446.

41 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing (SICOMP)*, 24(6):1235–1258, 1995.

42 Bernard M. E. Moret and Henry D. Shapiro. *Algorithms from P to NP (Vol. 1): Design and Efficiency*. Benjamin-Cummings Publishing Co., Inc., USA, 1991.

43 Juan L. Reutter, Miguel Romero, and Moshe Y. Vardi. Regular queries on graph databases. *Theory of Computing Systems (ToCS)*, 61(1):31–83, 2017.

44 Miguel Romero, Pablo Barceló, and Moshe Y. Vardi. The homomorphism problem for regular graph patterns. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005106.

45 Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Record*, 44(1):10–17, 2015.

46 Ryan Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 1227–1237, 2004. doi:10.1007/978-3-540-27836-8_101.

47 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

48 Virginia Vassilevska Williams. Algorithms column: An overview of the recent progress on matrix multiplication. *SIGACT News*, 43(4):57–59, 2012. doi:10.1145/2421119.2421134.

49 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

50 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 17–29, 2015. doi:10.4230/LIPIcs.IPEC.2015.17.

51 Virginia Vassilevska Williams. Some open problems in fine-grained complexity. *SIGACT News*, 49(4):29–35, 2018. doi:10.1145/3300150.3300158.

52 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

53 Peter T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, 2012. doi:10.1145/2206869.2206879.

54 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.

Ranked Enumeration of MSO Logic on Words

Pierre Bourhis
CNRS Lille, CRIStAL UMR 9189, University of Lille, INRIA Lille, France

Alejandro Grez
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile

Louis Jachiet
LTCI, IP Paris, France

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile

Abstract

In the last years, enumeration algorithms with bounded delay have attracted a lot of attention for several data management tasks. Given a query and the data, the task is to preprocess the data and then enumerate all the answers to the query one by one and without repetitions. This enumeration scheme is typically useful when the solutions are treated on the fly or when we want to stop the enumeration once the pertinent solutions have been found. However, with the current schemes, there is no restriction on the order how the solutions are given and this order usually depends on the techniques used and not on the relevance for the user.

In this paper we study the enumeration of monadic second order logic (MSO) over words when the solutions are ranked. We present a framework based on *MSO cost functions* that allows to express MSO formulae on words with a cost associated with each solution. We then demonstrate the generality of our framework which subsumes, for instance, document spanners and adds ranking to them. The main technical result of the paper is an algorithm for enumerating all the solutions of formulae in increasing order of cost efficiently, namely, with a linear preprocessing phase and logarithmic delay between solutions. The novelty of this algorithm is based on using functional data structures, in particular, by extending functional Brodal queues to suit with the ranked enumeration of MSO on words.

2012 ACM Subject Classification Theory of computation → Data structures and algorithms for data management; Theory of computation → Complexity theory and logic; Theory of computation → Formal languages and automata theory

Keywords and phrases Persistent data structures, Query evaluation, Enumeration algorithms

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.20

Funding *Pierre Bourhis*: Partially funded by the DeLTA project (ANR-16-CE40-0007).
Alejandro Grez: Partially funded by the Millennium Institute for Foundational Research on Data.
Cristian Riveros: Partially funded by the Millennium Institute for Foundational Research on Data.

1 Introduction

Managing and querying word structures such as texts has been one of the classical problems of different communities in computer science. In particular, this problem has been predominant in information extraction where the goal is to extract some subparts of a text. A logical approach that has brought a lot of attention in the database community is document spanners [13]. This logical framework provides a language for extracting subparts of a document. More specifically, regular spanners are based on regular expressions that fill relations with tuples of the texts' subparts. These relations can afterwards be queried by conjunctive or datalog-like queries.

The document spanners' main algorithmic problem is the efficient evaluation of a spanner over a word. Recently, a novel approach has been to focus on the enumeration problem to obtain efficient evaluation algorithms. The principle of an enumeration algorithm is to create a representation of the set of answers efficiently depending only on the input word's size and the query, and not in the number of answers. This time is called *the preprocessing time*. The second part of an enumeration algorithm is to enumerate the outputs one by one using the previous representation. The time between two consecutive outputs is called the *delay*. As for the preprocessing time, an efficient delay should not depend on the number of outputs, but only on the input size (i.e., word and query). In general, the most efficient enumeration algorithms have linear preprocessing time and constant delay, both in data complexity (i.e. in the size of the input word).

Several people have studied the enumeration problem over words following different formalisms. For example, [3, 6, 23] studied the enumeration problem for MSO logic, [14, 2] for regular spanners (i.e., automata), and [18] for streaming evaluation in complex event processing. For all these formalisms, it is shown that there exists an enumeration algorithm with linear time preprocessing and delay constant both in data complexity (i.e. in the size of the input word).

The interest of an efficient enumeration algorithm is to provide a process that can quickly give the first answers. Unfortunately, these answers may not be relevant for the user; that is, the enumeration process does not assume how the output will be ordered. A classical manner of considering the user's preferences is to associate a score to each solution and then rank them following this score; for instance one could ask for the matches ranked by order of length, or by the number of times a second pattern appears within the match. This approach of "scoring" solutions has been used particularly in the context of information extraction. Indeed, there have been several recent proposals [9, 8] to extend document spanners with annotations from a semi-ring. The proposed annotations are typically useful to capture the confidence of each solution [9]. For instance, [8] proves that the enumeration of the answers following their scores' order is possible with polynomial-time preprocessing and polynomial delay.

In this paper, we are interested in establishing a framework for scoring outputs and improve the bounds proved in [9]. We propose using what we called MSO cost functions, which are formulas in weighted logics [11] extended with open variables. These formulas provide a simple formalism for defining the output and scoring with MSO logic. We show that one can translate each MSO cost function to a cost transducer. These machines are a restricted form of weighted functional vset-automaton [9], for which there exists at most one run for any word and any valuation. We use cost transducers to study the ranked enumeration problem: enumerate all outputs in increasing rank order. Specifically, the main result of the paper is an algorithm for enumerating all the solutions of a cost transducer in increasing order efficiently; specifically, with a preprocessing phase linear in the input word and a logarithmic delay between solutions.

Our approach generalizes an algorithm for enumerating solutions proposed in [18, 14]. The preprocessing part builds a heap containing the answers with their score, and one step of the enumeration is simply a pop of the heap. For this, we use a general data structure that we called Heap of Words (HoW), having the classical heap operations of finding/deleting the minimal element, adding an element, and melding two heaps. We also need to add two new operations that allow us to concatenate a letter to and increase the score of all elements of the heap. Finally, we require that this structure is fully-persistent [10], i.e., that each of the previous operations returns a new heap without changing the previous one. To obtain the

required efficiency, we rely on a classical persistent data structure called Brodal queue that we extend in order to capture the new operations over the stored words and scores presented above. We call this extension an incremental Brodal queue.

Finally, for ranked query evaluation, there has been recent progress in the context of conjunctive queries: on the efficient computation of top-k queries [25] and the efficient ranked enumeration [24, 7]. These advances consider relational data (which is more general than words) and conjunctive queries (which is more restricted than MSO queries); they are thus incomparable to our work. However, it is important to note some similarities with our work, such as the need for an "advanced" priority queue (the Fibonacci heap [7]), which means that our incremental queues might be of great interest there.

Contributions. The contributions of this paper are threefold: **(i)** we introduce MSO cost functions, a framework to express MSO queries and scores, generalizing the proposals of document spanners; **(ii)** we give a ranked enumeration scheme that has linear preprocessing time and logarithmic delay in data complexity with a polynomial combined complexity; **(iii)** we introduce two new data structures for our scheme: the Heaps of Words and the incremental Brodal queues. Both of these structures might be of interest in other ranked enumerations schemes.

Organization. Section 2 introduces the ranked enumeration problem for MSO queries on words. Section 3 presents MSO cost functions and state the main result. In Section 4 we show an application of the main result to the setting of document spanners. Section 5 describes our enumeration scheme that rely on two data structures: the Heap of Words described in Section 6 and the incremental Brodal queues presented in Section 7. We finish with some conclusions in Section 8.

2 Preliminaries

Words. We denote by Σ a finite alphabet, Σ^* all words over Σ, and ϵ the empty-word of 0 length. Give a word $w = a_1 \dots a_n$, we write $w[i] = a_i$. For two words $u, v \in \Sigma^*$ we write $u \cdot v$ as the concatenation of u and v. We denote by $[n] = \{1, \dots, n\}$.

Ordered groups. A group is a pair $(\mathbb{G}, \oplus, \mathbb{0})$ where $\mathbb{G}$ is a set of elements, $\oplus$ is a binary operation over $\mathbb{G}$ that is associative, $\mathbb{0} \in \mathbb{G}$ is a neutral element for $\oplus$ (i.e., $\mathbb{0} \oplus g = g \oplus \mathbb{0} = g$) and every $g \in \mathbb{G}$ has an inverse with respect to $\oplus$ (i.e., $g \oplus g^{-1} = \mathbb{0}$ for some $g^{-1} \in \mathbb{G}$). A group is abelian if, in addition, $\oplus$ is commutative (i.e., $g_1 \oplus g_2 = g_2 \oplus g_1$). From now on, we assume that all groups are abelian. We say that $(\mathbb{G}, \oplus, \mathbb{0}, \preceq)$ is an ordered group if $(\mathbb{G}, \oplus, \mathbb{0})$ is a group and $\preceq$ is a total order over $\mathbb{G}$ that respects $\oplus$, namely, if $g_1 \preceq g_2$ then $g_1 \oplus g \preceq g_2 \oplus g$ for every $g, g_1, g_2 \in \mathbb{G}$. Examples of (abelian) ordered groups are $(\mathbb{Z}, +, 0, \leq)$ and $(\mathbb{Z}^k, +, (0, \dots, 0), \leq_k)$ where $\leq_k$ represents the lexicographic order over $\mathbb{Z}^k$.

MSO. We use monadic second-order logic for defining properties over words. As usual, we encode words as logical structures with an order predicate and unary predicates to represent the order and the letters of each positions of the word, respectively. More formally, fix an alphabet Σ and let $w \in \Sigma^*$ be a word of length n. We encode w as a structure $([n], \leq, (P_a)_{a \in \Sigma})$ where $[n]$ is the domain, $\leq$ is the total order over $[n]$, and $P_a = \{i \mid w[i] = a\}$. By some abuse of notation, we also use w to denote its corresponding logical structure.

A MSO-formula φ over Σ is given by:

$$\varphi \;:=\; x \leq y \;\mid\; P_a(x) \;\mid\; x \in X \;\mid\; \varphi \wedge \varphi \;\mid\; \neg\varphi \;\mid\; \exists x.\, \varphi \;\mid\; \exists X.\, \varphi$$

where $a \in \Sigma$, x and y are first-order (FO) variables, and X is a monadic second order (MSO) variable (i.e., a set variable). We write $\varphi(\bar{x}, \bar{X})$ where $\bar{x}$ and $\bar{X}$ are the sets of free FO and MSO variables of φ, respectively. An assignment σ for w is a function $\sigma : \bar{x} \cup \bar{X} \to 2^{[n]}$ such that $|\sigma(x)| = 1$ for every $x \in \bar{x}$ (note that we treat FO variables as a special case of MSO variables). As usual, we denote by $\mathrm{dom}(\sigma) = \bar{x} \cup \bar{X}$ the domain of the function σ. Then we write $(w, \sigma) \models \varphi$ when σ is an assignment over w, $\mathrm{dom}(\sigma) = \bar{x} \cup \bar{X}$, and w satisfies $\varphi(\bar{x}, \bar{X})$ when each variable in $\bar{x} \cup \bar{X}$ is instantiated by σ (see [20]). Given a formula $\varphi(\bar{x}, \bar{X})$, we define $[\![\varphi]\!](w) \;=\; \{\sigma \mid (w, \sigma) \models \varphi(\bar{x}, \bar{X})\}$. For the sake of simplification, from now on we will only use $\bar{X}$ to denote the free variables of $\varphi(\bar{X})$ and use $X \in \bar{X}$ for an FO or MSO variable.

For any assignment σ over w, we define the support of σ, denoted by $\mathrm{supp}(\sigma)$, as the set of positions mentioned in σ; formally, $\mathrm{supp}(\sigma) = \{i \mid \exists v \in \mathrm{dom}(\sigma).\, i \in \sigma(v)\}$. Furthermore, we encode assignments as sequences over the support as follows. Let $\mathrm{supp}(\sigma) = \{i_1, \ldots, i_m\}$ such that $i_j < i_{j+1}$ for every $j < m$. Then we define the (word) encoding of σ as:

$$\mathrm{enc}(\sigma) \;=\; (\bar{X}_1, i_1)(\bar{X}_2, i_2) \ldots (\bar{X}_m, i_m)$$

such that $\bar{X}_j = \{X \in \mathrm{dom}(\sigma) \mid i_j \in \sigma(X)\}$ for every $j \leq m$. That is, we represent σ as an increasing sequence of positions, where each position is labeled with the variables of σ where it belongs. This is the standard encoding used to represent assignments for running algorithms regarding MSO formulas [3, 6]. Finally, we define the size of σ as $|\,\mathrm{enc}(\sigma)| = |\,\mathrm{dom}(\sigma)| \cdot m$.

Enumeration algorithms. Given a formula φ and a word w, the main goal of the paper is to study the enumeration of assignments in $[\![\varphi]\!](w)$. We give here a general definition of enumeration algorithm and how we measure its delay. Later we use this to define the ranked enumeration problem of MSO.

As it is standard in the literature [3, 6, 23], we consider algorithms on Random Access Machines (RAM) with uniform cost measure [1] equipped with addition and subtraction as basic operations. A RAM has read-only input registers (containing the input I), read-write work memory registers and write-only output registers. We assume that group elements can be stored in constant space and that all group-related operations (i.e., to evaluate $g_1 \oplus g_2$, $g_1 \preceq g_2$ and g^{-1}) take constant time. We say that an algorithm $\mathcal{E}$ is an enumeration algorithm for MSO evaluation if $\mathcal{E}$ runs in two phases, for every MSO-formula φ and a word w.

1. The first phase, called the preprocessing phase, does not produce output, but may prepare data structures for use in the next phase.
2. The second phase, called the enumeration phase, occurs immediately after the precomputation phase. During this phase, the algorithm:
   - writes $\#\,\mathrm{enc}(\sigma_1)\#\,\mathrm{enc}(\sigma_2)\#\ldots\#\,\mathrm{enc}(\sigma_k)\#$ to the output registers where $\#$ is a distinct separator symbol, and σ_1, …, σ_k is an enumeration (without repetition) of the assignments of $[\![\varphi]\!](w)$;
   - it writes the first $\#$ as soon as the enumeration phase starts,
   - it stops immediately after writing the last $\#$.

The separation of $\mathcal{E}$'s operation into a preprocessing and enumeration phase is done to be able to make an output-sensitive analysis of $\mathcal{E}$'s complexity. Formally, we say that $\mathcal{E}$ has preprocessing time $f : \mathbb{N}^2 \to \mathbb{N}$ if there exists a constant C such that the number of instructions that $\mathcal{E}$ executes during the preprocessing phase on input (φ, w) is at most $C \times f(|\varphi|, |w|)$ for every MSO-formula φ and word w. Furthermore, we measure the delay as follows. Let

$\text{time}_i(\varphi, w)$ denote the time in the enumeration phase when the algorithm writes the i-th # (if it exists) when running on input (φ, w). Define $\text{delay}_i(\varphi, w) = \text{time}_{i+1}(\varphi, w) - \text{time}_i(\varphi, w)$. Further, let $\text{output}_i(\varphi, w)$ denote the i-th element that is output by $\mathcal{E}$ when running on input (φ, w), if it exists. We say that $\mathcal{E}$ has delay $g : \mathbb{N}^2 \to \mathbb{N}$ if there exists a constant D such that, for all φ and w, it holds that:

$$\text{delay}_i(\varphi, w) \;\leq\; D \times |\text{output}_i(\varphi, w)| \times g(|\varphi|, |w|)$$

for every $i \leq |[\![\varphi]\!](w)|$. Furthermore, if $[\![\varphi]\!](w)$ is empty, then $\text{delay}_1(\varphi, w) \leq k$, namely, it ends in constant time. Finally, we say that $\mathcal{E}$ has preprocessing time $f : \mathbb{N} \to \mathbb{N}$ and delay $g : \mathbb{N} \to \mathbb{N}$ in *data complexity*, if there exists a function $c : \mathbb{N} \to \mathbb{N}$ such that $\mathcal{E}$ has preprocessing time $c(|\varphi|) \times f(|w|)$ and has delay $c(|\varphi|) \times g(|w|)$ (i.e., f and g describe the complexity once φ is considered as fixed).

It is important to notice that, although we fix a particular encoding for assignments and we restrict the enumeration algorithms to this encoding, we can use any encoding for the assignments whenever there exists a linear transformation between $\text{enc}(\cdot)$ and the new encoding. Given the definition of delay, if we use an encoding $\text{enc}'(\sigma)$ for σ, and there exists a linear time transformation between $\text{enc}(\sigma)$ and $\text{enc}'(\sigma)$ for every σ, then the same enumeration algorithm works for $\text{enc}'(\cdot)$. In particular, whenever the encoding depends linearly over $\text{supp}(\sigma)$ and $|\bar{x} \cup \bar{X}|$, then the aforementioned property holds.

Ranked enumeration. For an MSO formula φ and $w \in \Sigma^*$, we consider the ranked enumeration of the set $[\![\varphi]\!](w)$. For this, we need to assign an order to the outputs and we do this by mapping each element to a total order set. Fix a set C with a total order $\preceq$ over C. A cost function is any partial function κ that maps words $w \in \Sigma^*$ and assignments σ to elements in C. Without loss of generality, we assume that κ is defined only over pairs (w, σ) such that σ is an assignment over w.

Let φ be an MSO formula and κ a cost function over $(C, \preceq)$. We define the ranked enumeration problem of (φ, κ) as

Problem:	RANK-ENUM$[\varphi, \kappa]$
Input:	A word $w \in \Sigma^*$.
Output:	Enumerate all $\sigma_1, \ldots, \sigma_k \in [\![\varphi]\!](w)$ without repetitions and such that $\kappa(w, \sigma_i) \preceq \kappa(w, \sigma_{i+1})$.

Note that we consider the version of the problem in data-complexity where φ and κ are fixed. We say that RANK-ENUM$[\varphi, \kappa]$ can be solved with preprocessing time $f(n)$ and delay $g(n)$ if there exists an enumeration algorithm $\mathcal{E}$ that runs with preprocessing time $f(n)$ and delay $g(n)$ and, for every $w \in \Sigma^*$, $\mathcal{E}$ enumerates $[\![\varphi]\!](w)$ in increasing ordered according to κ. In the next section, we give a language to define cost functions and we state our main result.

3 MSO cost functions

To state our main result about ranked enumeration of MSO, first we need to choose a formalism to define cost functions. We do this by staying in the same setting of MSO logic by considering weighted logics over words [11, 12, 19]. Functions defined by extensions of MSO has been studied by using weighted automata, but also people have found it counterparts by extending MSO with a semiring. We use here a fragment of weighted MSO parametrized by an ordered group to fit our purpose.

Fix an ordered group $(\mathbb{G}, \oplus, \mathbb{0}, \preceq)$. A weighted MSO-formula α over Σ and $\mathbb{G}$ is given by the following syntax:

$$\alpha \;:=\; [\varphi \mapsto g] \;\mid\; \alpha \oplus \alpha \;\mid\; \Sigma\, x.\, \alpha$$

where φ is an MSO-formula, $g \in \mathbb{G}$, and x is an FO variable. Further, we assume that the $\Sigma\, x$ quantifier cannot be nested. For example, $(\Sigma\, x.[\varphi \mapsto g]) \oplus (\Sigma\, y.[\varphi' \mapsto g'])$ is a valid formula but $\Sigma\, x.\Sigma\, y.[\varphi \mapsto g]$ is not. Similar than for MSO formulas, we write $\alpha(\bar{x}, \bar{X})$ to state explicitly the sets of FO-variables $\bar{x}$ and of MSO variables $\bar{X}$ that are free in α.

Let σ be an assignment. For any FO-variable x and $i \in \mathbb{N}$ we denote by $\sigma[x \to i]$ the extension of σ with x assigned to i, namely, $\mathrm{dom}(\sigma[x \to i]) = \{x\} \cup \mathrm{dom}(\sigma)$ such that $\sigma[x \to i](x) = \{i\}$ and $\sigma[x \to i](y) = \sigma(y)$ for every $y \in \mathrm{dom}(\sigma) \setminus \{x\}$. We define the semantics of a weighted MSO formula α as a function from words and assignments to elements in $\mathbb{G}$. Formally, for every $w \in \Sigma$ and every assignment σ over w we define the output $[\![\alpha]\!](w, \sigma)$ recursively as follows:

$$[\![[\varphi \mapsto g]]\!](w,\sigma) = \begin{cases} g & \text{if } (w,\sigma) \models \varphi \\ \mathbb{0} & \text{otherwise.} \end{cases} \qquad \begin{aligned} [\![\alpha \oplus \alpha']\!](w,\sigma) &= [\![\alpha]\!](w,\sigma) \oplus [\![\alpha']\!](w,\sigma) \\ [\![\Sigma\, x.\, \alpha]\!](w,\sigma) &= \bigoplus_{i=1}^{|w|} [\![\alpha]\!](w, \sigma[x \to i]) \end{aligned}$$

where φ is any MSO-formula, α and α' are weighted MSO formulas, and $g \in \mathbb{G}$. By some abuse of notation, in the following we will not make distinction between α and $[\![\alpha]\!]$, that is, the cost function over $\mathbb{G}$ defined by α.

▶ **Example 1.** Consider the alphabet $\{a, b\}$ and suppose that we want to define a cost function that counts the number of a-letters between two variables x and y. This can be defined in weighted MSO over $\mathbb{Z}$ as follows:

$$\alpha_1 \;:=\; \Sigma\, z.\big[(x \leq z \wedge z \leq y \wedge P_a(z)) \mapsto 1\big]$$

Here, α_1 uses z to count over all positions of the word and we count 1 whenever z is labeled with a and is between x and y, and we count 0, otherwise, which is the identity of $\mathbb{Z}$.

▶ **Example 2.** Consider again the alphabet $\{a, b\}$ and suppose that we want a cost function to compare assignments over variables (x, y) lexicographically. For this, we can write a weighted MSO-formula over $\mathbb{Z}^2$ that maps each assignment σ over x and y to a pair $(\sigma(x), \sigma(y))$. This can be defined in weighted MSO over $\mathbb{Z}^2$ as follows:

$$\alpha_2 \;:=\; \big(\Sigma\, z_1.\,\big[(z_1 \leq x) \mapsto (1,0)\big]\big) + \big(\Sigma\, z_2.\,\big[(z_2 \leq y) \mapsto (0,1)\big]\big)$$

Similar than for the previous example, we use the Σ-quantifier to add in the first and second component the value of x and y, respectively. In fact, for every assignment $\sigma = \{x \to i, y \to j\}$ over $w \in \Sigma^*$ it holds that $[\![\alpha_2]\!](w, \sigma) = (i, j)$.

Strictly speaking, the syntax and semantics of weighted MSO defined above is a restricted version of weighted logics [11], in the sense that weighted logics is usually defined over a semiring, which has two binary operations $\oplus$ and $\odot$. Indeed, it will be interesting to have a better understanding of the expressibility of MSO cost functions, or to extend our results for weighted logics over semiring. We leave this for future work.

We are ready to state the main result of the paper about ranked enumeration of MSO.

▶ **Theorem 3.** *Fix an alphabet Σ and an ordered group $\mathbb{G}$. For every MSO formula φ over Σ and every weighted MSO formula α over Σ and $\mathbb{G}$ the problem RANK-ENUM$[\varphi, \alpha]$ can be solved with linear preprocessing time and logarithmic delay.*

As it is common for MSO logic over words, we prove this result by developing an enumeration algorithm using automata theory. Specifically, we define a weighted automata model, that we called cost transducer, and show that its expressiveness is equivalent to the combination of (boolean) MSO and weighted MSO logic.

From now on, fix an input alphabet Σ and an output alphabet Γ. Furthermore, fix an ordered group $(\mathbb{G}, \oplus, \mathbb{0}, \preceq)$. A *cost transducer* over $\mathbb{G}$ is a tuple $\mathcal{T} = (Q, \Delta, \kappa, I, F)$, where Q is the set of states, $\Delta \subseteq Q \times \Sigma \times 2^{\Gamma} \times Q$ is the transition relation, $\kappa : \Delta \to \mathbb{G}$ is a function that associates a cost to every transition of Δ, and $I : Q \to \mathbb{G}$, $F : Q \to \mathbb{G}$ are partial functions that associate a cost in $\mathbb{G}$ to (some) states in Q. The functions I and F are partial functions because they naturally define the set of initial and final states as $\mathrm{dom}(I)$ and $\mathrm{dom}(F)$, respectively. A run of $\mathcal{T}$ over a word $w = a_1 a_2 \dots a_n$ is a sequence of transitions $\rho : q_0 \xrightarrow{a_1/\bar{X}_1} q_1 \xrightarrow{a_2/\bar{X}_2} \dots \xrightarrow{a_n/\bar{X}_n} q_n$ such that $q_0 \in \mathrm{dom}(I)$ and $(q_{i-1}, a_i, \bar{X}_i, q_i) \in \Delta$ for every $i \leq n$. We say that ρ is accepting if $q_n \in \mathrm{dom}(F)$.

For a run ρ as defined above, let $\{i_1, \dots, i_m\} \subseteq [n]$ be all the positions of ρ such that $\bar{X}_{i_j} \neq \emptyset$ and $i_j < i_{j+1}$ for all $j \leq m$. Then we define the output of ρ as the sequence:

$$\mathrm{out}(\rho) \;=\; (\bar{X}_{i_1}, i_1)(\bar{X}_{i_2}, i_2) \dots (\bar{X}_{i_m}, i_m)$$

Moreover, we extend κ over accepting runs ρ by adding the costs of all transitions of ρ plus the initial and final cost, namely:

$$\kappa(\rho) = I(q_0) \oplus \bigoplus_{i=1}^{|w|} \kappa((q_{i-1}, a_i, \bar{X}_i, q_i)) \oplus F(q_n).$$

Note that $\mathrm{out}(\rho)$ defines the encoding of some assignment σ over w with $\mathrm{dom}(\sigma) = \Gamma$ and $\mathrm{out}(\rho) = \mathrm{enc}(\sigma)$. Of course, the opposite direction is not true: for some assignment σ there could be no run ρ that defines σ and, moreover, there could be two runs ρ_1 and ρ_2 such that $\mathrm{out}(\rho_1) = \mathrm{out}(\rho_2) = \mathrm{enc}(\sigma)$, but $\kappa(\rho_1) \neq \kappa(\rho_2)$. For this reason, we impose an additional restriction to cost transducers: we assume that all cost transducers in this paper are unambiguous, that is, for every $w \in \Sigma^*$ there does not exist two distinct runs ρ_1 and ρ_2 of w such that $\mathrm{out}(\rho_1) = \mathrm{out}(\rho_2)$. In other words, a cost transducers satisfies that for every $w \in \Sigma^*$ and assignment σ there exists at most one run ρ such that $\mathrm{out}(\rho) = \mathrm{enc}(\sigma)$.

Given the unambiguous restriction of cost transducers, we can define a partial function from pairs (w, σ) to $\mathbb{G}$ as $\mathrm{cost}_{\mathcal{T}}(w, \sigma) = \kappa(\rho)$ whenever there exists a run ρ of w such that $\mathrm{out}(\rho) = \mathrm{enc}(\sigma)$. Otherwise $\mathrm{cost}_{\mathcal{T}}(w, \sigma)$ is not defined. Given that for some pairs (w, σ) the function $\mathrm{cost}_{\mathcal{T}}$ is not defined, we can define the set $[\![\mathcal{T}]\!](w) = \{\sigma \mid \mathrm{cost}_{\mathcal{T}}(w, \sigma) \text{ is defined}\}$ of all outputs of $\mathcal{T}$ over w.

It is important to notice that, given $w \in \Sigma^*$, a cost transducer $\mathcal{T}$ is in charge of (1) defining the set of assignments $[\![\mathcal{T}]\!](w)$ and (2) assigning a cost $[\![\mathcal{T}]\!](w, \sigma)$ for each output $\sigma \in [\![\mathcal{T}]\!](w)$. These two tasks are separated in our setting of ranked MSO enumeration by having a MSO formula φ that defines the outputs $[\![\varphi]\!]$ and a weighted MSO formula α to assign a cost to each pair (w, σ). In fact, one can show that cost transducers are equally expressive than combining MSO plus weighted MSO.

▶ **Proposition 4.** *For every cost transducer $\mathcal{T}$, there exists a MSO formula $\varphi_{\mathcal{T}}$ and weighted MSO formula $\alpha_{\mathcal{T}}$ such that $[\![\mathcal{T}]\!] = [\![\varphi_{\mathcal{T}}]\!]$ and $\mathrm{cost}_{\mathcal{T}}(w, \sigma) = [\![\alpha_{\mathcal{T}}]\!](w, \sigma)$ for every $\sigma \in [\![\mathcal{T}]\!](w)$. Moreover, for every MSO formula φ and weighted MSO formula α, there exists a cost transducer $\mathcal{T}_{\varphi,\alpha}$ such that $[\![\varphi]\!] = [\![\mathcal{T}_{\varphi,\alpha}]\!]$ and $[\![\alpha]\!](w, \sigma) = \mathrm{cost}_{\mathcal{T}_{\varphi,\alpha}}(w, \sigma)$ for every $\sigma \in [\![\varphi]\!](w)$.*

By the previous result, we can represent pairs of formulas (φ, α) by using cost transducers and vice-versa. Similar than for MSO [22], there exists a non-elementary blow-up for going from (φ, α) to a cost transducer and this blow-up cannot be avoided [16].

To solve the problem RANK-ENUM$[\varphi, \alpha]$ we can use a cost transducer $\mathcal{T}_{\varphi,\alpha}$ to enumerate all its outputs following the cost assigned by this machine. More concretely, we study the following rank enumeration problem for cost transducers:

Problem:	RANK-ENUM-T
Input:	A cost transducer $\mathcal{T}$ and a word $w \in \Sigma^*$.
Output:	Enumerate all $\sigma_1, \ldots, \sigma_k \in [\![\mathcal{T}]\!](w)$ without repetitions and such that $\text{cost}_{\mathcal{T}}(w, \sigma_i) \preceq \text{cost}_{\mathcal{T}}(w, \sigma_{i+1})$.

Note that for RANK-ENUM-T we consider the cost transducer as part of the input[1]. Indeed, for this case we can provide an enumeration algorithm with stronger guarantees regarding the preprocessing time in terms of $\mathcal{T}$. We now give the theorem formalizing the main result of this paper, which will be proven in the next section:

▶ **Theorem 5.** *The problem RANK-ENUM-T can be solved with $|\mathcal{T}| \cdot |w|$ preprocessing time and $\log(|\mathcal{T}| \cdot |w|)$-delay.*

In the rest of the paper, we present the above mentioned ranked enumeration algorithm. We start by showing a general algorithm based on a novel data structure called a Heap of Words. In Section 6, we provide the implementation of this structure. In Section 7, we show how to implement the incremental Brodal queues, a technical data structure needed to obtain the required efficiency. Before presenting the technical details of this algorithm, in the next section we show an application of this result in the framework of document spanners [13, 9].

4 Application: document spanners

The framework of document spanners was proposed in [13] as a formalization of rule-based information extraction and has attracted a lot of attention both in terms of the formalism [15, 21] and the enumeration problem associated to it [14]. Recently, an extension of document spanners has been proposed to enhance the extraction process with annotations [9, 8]. These annotations serve as auxiliary information of the extracted data such as confidence, support, or confidentiality measures. To extend spanners, this framework follows the approach of provenance semiring by annotating the output with elements from a semiring and propagating the annotations by using the semiring operators. Next we give the core definitions of [9] and we state the application of our main results to this setting.

We start by defining the central elements of document spanners: documents and spans. Fix a finite alphabet Σ. A document over Σ (or just a document) is a string $d = a_1 \ldots a_n \in \Sigma^*$ and a span is a pair $s = [i, j\rangle$ with $1 \leq i \leq j \leq n + 1$. A span represents a continuous region of d, whose content is the substring from positions i to $j - 1$. Formally, the content of span $[i, j\rangle$ is defined as $d[i, j\rangle = a_i \ldots a_{j-1}$; if $i = j$, then $d[i, i\rangle = \epsilon$. Fix a finite set of variables $\mathbf{X}$. A mapping μ over d is a function from $\mathbf{X}$ to the spans of d. A document spanner (or just spanner) is a function that maps each document d to a set of mappings over d.

To annotate mappings, we need to introduce semirings. A semiring $(K, \oplus, \odot, \mathtt{0}, \mathtt{1})$ is an algebraic structure where K is a non-empty set, $\oplus$ and $\odot$ are binary operations over K, and $\mathtt{0}, \mathtt{1} \in K$. Furthermore, $\oplus$ and $\odot$ are associative, $\mathtt{0}$ and $\mathtt{1}$ are the identities of $\oplus$ and $\odot$ respectively, $\oplus$ is commutative, $\odot$ distributes over $\oplus$, and $\mathtt{0}$ annihilates K (i.e.,

[1] In Section 2 we introduce the setting of ranked enumeration for MSO formulas and cost functions. One can easily extend this setting and the definiton of enumeration algorithms for cost transducer.

$\mathtt{0} \odot k = k \odot \mathtt{0} = \mathtt{0}$ for all $k \in K$). We will use $\bigoplus_X$ or $\bigodot_X$ for the $\oplus$- or $\odot$-iteration over all elements in some set X, respectively. An ordered semiring $(K, \oplus, \odot, \mathtt{0}, \mathtt{1}, \preceq)$ is a semiring extended with a total order $\preceq$ over K such that $\preceq$ preserves $\oplus$ and $\odot$, namely, $k_1 \preceq k_2$ implies $k_1 * k \preceq k_2 * k$ for $* \in \{\oplus, \odot\}$. From now on, we will assume that all semirings are ordered. A semifield [17] is a semiring $(K, \oplus, \odot, \mathtt{0}, \mathtt{1})$ where each $k \in K \setminus \{\mathtt{0}\}$ has a multiplicative inverse (i.e., $(K \setminus \{\mathtt{0}\}, \odot, \mathtt{1})$ forms a group). Examples of ordered semifields are the tropical semiring $(\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0, \leq)$ and the semiring of non-negative rational numbers $(\mathbb{Q}_{\geq 0}, +, \times, 0, 1, \leq)$.

Fix a semiring $(K, \oplus, \odot, \mathtt{0}, \mathtt{1})$. Let $\mathbf{X}$ be a set of variables and define $\mathcal{C}(\mathbf{X})$ as the set of captures $\{x\vdash, \dashv x \mid x \in \mathbf{X}\}$. To extend spanners with annotations, we use the formalism of weighted variable set automata [9] which defines the class of all regular spanners with annotations, also called regular annotators. A weighted variable set automaton (wVA) over K is a tuple $\mathcal{A} = (\mathbf{X}, Q, \delta, I, F)$ such that $\mathbf{X}$ is a finite set of variables, Q is a finite set of states, $\delta : Q \times (\Sigma \cup \mathcal{C}(\mathbf{X})) \times Q \to K$ is a weighted transition function and $I : Q \to K$ and $F : Q \to K$ are the initial and final weight functions, respectively. A run ρ over a document $d = a_1 \cdots a_n$ is a sequence of the form:

$$\rho \;:=\; (q_0, i_0) \xrightarrow{o_1} (q_1, i_1) \xrightarrow{o_2} \cdots \xrightarrow{o_m} (q_m, i_m)$$

where (1) $1 = i_0 \leq i_1 \leq \cdots \leq i_m = n + 1$, (2) each $q_j \in Q$ with $I(q_0) \neq \mathtt{0} \neq F(q_m)$, (3) $\delta(q_j, o_{j+1}, q_{j+1}) \neq \mathtt{0}$, and (4) $i_{j+1} = i_j$ if $o_{j+1} \in \mathcal{C}(\mathbf{X})$ and $i_{j+1} = i_j + 1$ otherwise. In addition, we say that a run ρ is valid if for every $x \in \mathbf{X}$ there exist exactly one index i with $o_i = x\vdash$, exactly one index j with $o_j = \dashv x$, and $i < j$. We denote by $\mathrm{Run}_{\mathcal{A}}(d)$ the set of all valid runs of $\mathcal{A}$ over d. Note that for some wVA $\mathcal{A}$ and document d there could exist runs of $\mathcal{A}$ over d that are not valid. For this reason, we say that $\mathcal{A}$ is functional if every run ρ of $\mathcal{A}$ over d is valid for every document d. Given that some decision problems for non-functional variable-set automata are NP-hard [15, 21], from now on we assume that all wVA are functional.

A valid run ρ like above naturally defines a mapping μ^ρ over $\mathbf{X}$ that maps each x to the span $[i_j, i_{j'}\rangle$ where $o_{i_j} = x\vdash$ and $o_{i_{j'}} = \dashv x$. Furthermore, we associate a weight in K to ρ by multiplying all the weights of the transitions as follows:

$$W(\rho) \;:=\; I(q_0) \odot \bigodot_{j=1}^{m} \delta(q_j, o_{j+1}, q_{j+1}) \odot F(q_m).$$

We define the set of output mappings of $\mathcal{A}$ over d as $[\![\mathcal{A}]\!](d) = \{\mu^\rho \mid \rho \in \mathrm{Run}_{\mathcal{A}}(d)\}$. Given a mapping $\mu \in [\![\mathcal{A}]\!](d)$, we associate the weight $W_{\mathcal{A},d}(\mu) = \bigoplus_{\rho \in \mathrm{Run}_{\mathcal{A}}(d) : \mu = \mu^\rho} W(\rho)$. Intuitively, each $\mu \in [\![\mathcal{A}]\!](d)$ contains relevant data extracted by $\mathcal{A}$ from d, and $W_{\mathcal{A},d}(\mu)$ is the annotation attached to μ obtained during the extraction process, e.g. confidence or support.

In [9], the problem of ranked annotator enumeration was proposed, which for the sake of completeness we present next[2]:

Problem:	RA-ENUM
Input:	A wVA $\mathcal{A}$ over an ordered semiring K and a document d.
Output:	Enumerate all $\mu_1, \ldots, \mu_k \in [\![\mathcal{A}]\!](d)$ without repetitions and such that $W_{\mathcal{A},d}(\mu_1) \preceq W_{\mathcal{A},d}(\mu_{i+1})$.

[2] In [9] they considered positively ordered semiring, which is slightly more general that the notion of ordered semiring used here.

RA-ENUM was studied in [9] and an enumeration algorithm was provided with polynomial preprocessing and polynomial delay in terms of $|\mathcal{A}|$ and $|d|$. By using the framework of MSO cost functions, we can give a better algorithm for a special case of RA-ENUM. We say that a wVA $\mathcal{A}$ is unambiguous if, for every document d and $\mu \in [\![\mathcal{A}]\!](d)$, there exists at most one run $\rho \in \text{Run}_{\mathcal{A}}(d)$ such that $\mu = \mu^\rho$. The connection between cost transducers and wVA is direct, although the former works over groups and wVA works over semirings. For this reason, we restrict wVA to semifields and give the following result.

▶ **Corollary 6.** *The problem RA-ENUM can be solved with $|\mathcal{A}| \cdot |d|$ preprocessing time and $\log(|\mathcal{A}| \cdot |d|)$-delay when $\mathcal{A}$ is unambiguous and K is an ordered semifield.*

Although the previous result is a restricted case of RA-ENUM and a direct consequence of Theorem 5, to the best of our knowledge this is the first non-trivial ranked enumeration algorithm proposed for the framework of document spanner.

5 Ranked enumeration algorithm

In this section, we will see how novel data structures can solve the ranked enumeration problem for cost transducers on words. We provide an algorithm for the RANK-ENUM-T problem, which uses a structure called *Heap of Words* (HoW) as a black box. We specify the interface of the HoW, to then present the full algorithm. The HoW structure is addressed in detail in the next section. This structure has the property of being fully-persistent. Given that this is a crucial property, we start with a brief introduction to this concept.

Fully-persistent data structures. A data structure is said *fully-persistent* [10] when no operation can modify the data structure. In a fully-persistent data structure, all the operations return new data structures, without changing the original ones. While this seems to be a restriction on the possible operations, it allows "sharing". For instance, with a fully-persistent linked list data structure, we can keep two lists l_1, l_2 with l_1 being some value followed by the content of l_2 and since no operation modifies the content of l_1 or l_2 there is no risk that an access to l_1 modifies indirectly l_2. In contrast, if we had allowed an operation that modifies the first value of a list in place (i.e., without returning a new list containing the modification), the applying this new operation on l_2 would have modified both l_1 and l_2.

All data structures that we use in this paper are fully-persistent. We use these data structures to store and enumerate the outputs of the cost transducer while, at the same time, share and modify the outputs without any risk of losing them. For more information of fully-persistent data structures, we refer the reader to [10].

The HoW data structure. A Heap of Words (HoW) over an ordered group $(\mathbb{G}, \oplus, \mathbb{0}, \preceq)$ is a data structure h that stores a finite set $\{[w_1 : g_1], \ldots, [w_n : g_n]\}$ where each $[w_i : g_i]$ is a pair composed by a word $w_i \in \Sigma^*$ and a priority $g_i \in \mathbb{G}$. Further, we assume that $w_i \neq w_j$ for every $i \neq j$, namely, the stored words form a set too. We define $[\![h]\!] = \{w_1, \ldots, w_n\}$ as the content of h and, given the previous restriction, there is a one-to-one correspondence between $[w_i : g_i]$ and w_i. Notice that we will usually write $h = \{[w_1 : g_1], \ldots, [w_n : g_n]\}$ to denote that h stores $[w_1 : g_1], \ldots, [w_n : g_n]$ but, strictly speaking, h is a data structure (i.e., a heap). Finally, we denote by $\emptyset$ the empty HoW.

The purpose of a HoW h is to store words and retrieve quickly the pair $[w : g]$ with minimum priority with respect to the order $\preceq$ of the group. We also want to manage h by *deleting* the word with minimum priority, *adding* new words, *increasing* the priority of

Algorithm 1 Preprocessing and enumeration phases for RANK-ENUM-T.

input: $\mathcal{T} = (Q, \Delta, \kappa, I, F)$ and $w = a_1 \dots a_n$.

1: **procedure** PREPROCESSING($\mathcal{T}, w$)
2: **for each** $q \in \mathrm{dom}(I)$ **do**
3: $h_q^0 \leftarrow \textsc{Add}(\emptyset, [\epsilon : I(q)])$
4: **for** i **from** 1 **to** n **do**
5: **for each** $t = (p, a_i, \bar{X}, q) \in \Delta$ **do**
6: $h \leftarrow h_p^{i-1}$
7: **if** $\bar{X} \neq \emptyset$ **then**
8: $h \leftarrow \textsc{ExtendBy}(h, (\bar{X}, i))$
9: $h \leftarrow \textsc{IncreaseBy}(h, \kappa(t))$
10: $h_q^i \leftarrow \textsc{Meld}(h_q^i, h)$
11: **for each** $q \in \mathrm{dom}(F)$ **do**
12: $h \leftarrow \textsc{IncreaseBy}(h_q^n, F(q))$
13: $h_{\text{out}} \leftarrow \textsc{Meld}(h_{\text{out}}, h)$
14: **return** h_{out}

input: A heap of words h.

1: **procedure** ENUMERATION(h)
2: `write` #
3: **while** $h \neq \emptyset$ **do**
4: `write` FINDMIN(h)
5: $h \leftarrow \textsc{DeleteMin}(h)$
6: `write` #

all elements by some $g \in \mathbb{G}$, or *extending* all words with a new letter $a \in \Sigma$. Furthermore, we want to build the union of two HoWs. More formally, we consider the following set of functions to manage HoWs. For HoWs h, h_1, and h_2, $w \in \Sigma^*$, $g \in \mathbb{G}$, and $a \in \Sigma$ we define:

$$\begin{array}{llll} w' := \textsc{FindMin}(h) & h' := \textsc{Meld}(h_1, h_2) & \text{s.t. } [\![h_1]\!] \cap [\![h_2]\!] = \emptyset \\ h' := \textsc{DeleteMin}(h) & h' := \textsc{Add}(h, [w : g]) & \text{s.t. } w \notin [\![h]\!] \\ h' := \textsc{IncreaseBy}(h, g) & h' := \textsc{ExtendBy}(h, a) & \end{array}$$

where h' is a new HoW and $w' \in \Sigma^*$. In general, each of such functions receives a HoW and outputs a HoW h'. As it was explained before, this data structure is fully-persistent and, therefore, after applying any of these functions, both the output h' and its previous version h are available. Now, we define the semantics of each operation. Let $h = \{[w_1 : g_1], \dots, [w_n : g_n]\}$. The FINDMIN of h returns a word w' such that $[w' : g']$ is stored in h and g' is minimal among all the priorities stored, formally, $[w' : g'] \in h$ and $g' = \min\{g \mid [w : g] \in h\}$. If there are several w' satisfying this property, one is picked arbitrarily. Operation DELETEMIN returns a new HoW h' that stores the set represented by h without the pair of the word returned by $\textsc{FindMin}(h)$, that is, $h' = h \setminus \{[w' : g']\}$ where $w' = \textsc{FindMin}(h)$ and $g' = \min\{g \mid [w : g] \in h\}$. Finally, the functions ADD, INCREASEBY, EXTENDBY, and MELD are formally defined as:

$$\begin{array}{ll} \textsc{Meld}(h_1, h_2) := h_1 \cup h_2 & \textsc{IncreaseBy}(h, g) := \{[w_1 : (g_1 \oplus g)], \dots, [w_n : (g_n \oplus g)]\} \\ \textsc{Add}(h, [w : g]) := h \cup \{[w : g]\} & \textsc{ExtendBy}(h, a) := \{[(w_1 \cdot a) : g_1], \dots, [(w_n \cdot a) : g_n]\} \end{array}$$

We assume that ADD, INCREASEBY, EXTENDBY and MELD take constant time and FINDMIN takes $\mathcal{O}(|w'|)$ where $w' = \textsc{FindMin}(h)$. For $\textsc{DeleteMin}(h)$, if h was built using n operations ADD, INCREASEBY, EXTENDBY and MELD followed by some number of operations DELETEMIN then computing $\textsc{DeleteMin}(h)$ takes $\mathcal{O}(|w'| \cdot \log(n))$ where $w' = \textsc{FindMin}(h)$. In the next section we show how to implement HoWs in order to satisfy these requirements. For now, we assume the existence of this data structure and use it to solve RANK-ENUM-T.

The algorithm. In Algorithm 1, we show the preprocessing phase and the enumeration phase to solve RANK-ENUM-T. On one hand, the PREPROCESSING procedure receives

a cost transducer $\mathcal{T} = (Q, \Delta, \kappa, I, F)$ and a word $w \in \Sigma^*$, and computes a HoW h_{out}. On the other hand, the ENUMERATION procedure receives a HoW (i.e., h_{out}) and enumerates $\text{enc}(\sigma_1), \ldots, \text{enc}(\sigma_k)$ such that $\{\sigma_1, \ldots, \sigma_k\} = [\![\mathcal{T}]\!](w)$ and $\text{cost}_{\mathcal{T}}(w, \sigma_i) \preceq \text{cost}_{\mathcal{T}}(w, \sigma_{i+1})$.

In both procedures we use HoW to compute the set of answers. Indeed, for each $q \in Q$ and each $i \in \{0, \ldots, |w|\}$ we compute a HoW h_q^i, and also compute a h_{out} to store the final results. We assume that all HoWs are empty (i.e., $h_{\text{out}} = \emptyset$ and $h_q^i = \emptyset$) when the algorithm starts. For each i, we call the set $\{h_q^i \mid q \in Q\}$ the i-level of HoW. Starting from the 0-level (lines 2-3), the preprocessing phase goes level by level, updating the i-level with the previous $(i-1)$-level (lines 4-10). It is important to note here that the MELD(h_q^i, h) call (line 10) is well-defined since $\mathcal{T}$ is unambiguous (i.e., $[\![h_q^i]\!] \cap [\![h]\!] = \emptyset$). After reaching the last n-level, the algorithm joins all HoWs $\{h_q^n \mid q \in \text{dom}(F)\}$ into h_{out}, by incrementing first their cost with $F(q)$ and melding them into h_{out} (lines 11-13). Finally, the preprocessing phase return h_{out} as output (line 14).

In order to understand the preprocessing algorithm, one has to notice that all the evaluation is based on a very simple fact. Let $w_i = a_1 \ldots a_i$ and define the set $\text{Run}_{\mathcal{T}}(q, w_i)$ of all partial runs of $\mathcal{T}$ over w_i that end in state q. For any of such runs $\rho = q_0 \xrightarrow{a_1/\bar{X}_1} \ldots \xrightarrow{a_i/\bar{X}_i} q_i \in \text{Run}_{\mathcal{T}}(q, w_i)$, define the partial cost of ρ as $\kappa^*(\rho) = I(q_0) \oplus \bigoplus_{j=1}^{i} \kappa((q_{j-1}, a_j, \bar{X}_j, q_j))$. After executing PREPROCESSING, it will hold that: $h_q^i = \big\{ [\text{out}(\rho) : \kappa^*(\rho)] \mid \rho \in \text{Run}_{\mathcal{T}}(q, w_i) \big\}$. This is certainly true for h_q^0 after lines 2-3 are executed. Then, if this is true for $(i-1)$-level, after the i-th iteration of lines 5-10 we will have that h_q^i contains all pairs of the form $[\text{out}(\rho) \cdot (\bar{X}, i) : \kappa^*(\rho) \oplus \kappa(t)]$ for each $t = (p, a_i, \bar{X}, q) \in \Delta$, plus all pairs $[\text{out}(\rho) : \kappa^*(\rho) \oplus \kappa(t)]$ for each $t = (p, a_i, \emptyset, q) \in \Delta$ and $\rho \in \text{Run}_{\mathcal{T}}(p, w_{i-1})$. Given that each line takes constant time, we can conclude that the preprocessing phase takes time $\mathcal{O}(|\mathcal{T}| \cdot |w|)$ as expected.

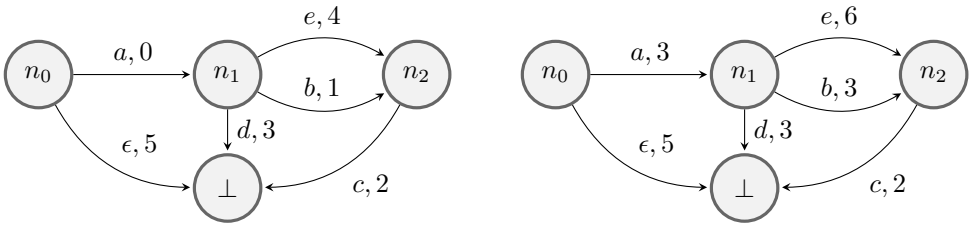
For the enumeration phase, we extract each output from h_{out}, one by one, by alternating between the FINDMIN and DELETEMIN procedures. Since with DELETEMIN we remove the minimum element of h after printing it, the correctness of the enumeration phase is straightforward. Notice that this enumeration will print all outputs in increasing order of priority. Furthermore, it will not print any output twice given that h_{out} contains no repetitions. To bound the time, notice that the number of ADD, INCREASEBY, EXTENDBY and MELD functions used during the pre-processing is at most $\mathcal{O}(|\mathcal{T}| \cdot |w|)$. For this reason, the delay between each output w' is bounded by $\mathcal{O}(\log(|\mathcal{T}| \cdot |w|) \cdot |w'|)$, satisfying the promised delay between outputs.

We want to finish this section by emphasizing that the ranked enumeration problem of cost transducers reduces to computing efficiently the HoW's methods. Moreover, it is crucial in this algorithm that this data structure is fully-persistent, and each operation takes constant time. Indeed, this allows us to pass the outputs between levels very efficiently and without losing the outputs of the previous levels.

6 The implementation of HoW data structure

In this section we focus on the HoW data structure and explain its implementation using yet another structure called incremental Brodal queue. We begin by explaining the general technique we use to store sets of strings with priorities, and end by giving a full implementation of the functions to manage HoWs.

Let Σ be a possibly infinite alphabet and $\mathbb{G} = (\mathbb{G}, \oplus, \mathbb{0}, \preceq)$ an order group. A *string-DAG* over Σ and $\mathbb{G}$ is a DAG $D = (V, E)$ where the edges are annotated with symbols in $\Sigma \cup \{\epsilon\}$ and priorities in $\mathbb{G}$. Formally, each edge has the form $e = (u, a, g, v)$, where $u, v \in V$, $a \in \Sigma \cup \{\epsilon\}$ and $g \in \mathbb{G}$. Given a path $\rho = v_1 \xrightarrow{a_1, g_1} \ldots \xrightarrow{a_k, g_k} v_k$, let $[w_\rho : g_\rho]$ be the pair defined by



(a) A string-DAG D. **(b)** The string-DAG prioritize(D).

Figure 1 A string-DAG D and the result of prioritize(D).

ρ, where $w_\rho = a_1 \dots a_k$ and $g_\rho = g_1 \oplus \dots \oplus g_k$. We make two more assumptions that any string-DAG must satisfy. First, we assume that there is a special sink vertex $\bot \in V$ that is reachable from any $v \in V$, has no outgoing edges, and that all edges with ϵ must point to $\bot$. Second, we assume that, for every $v \in V$ and every two different paths ρ and ρ' from v to $\bot$, it holds that $w_\rho \neq w_{\rho'}$. Given these two assumptions, we say that each $v \in V$ encodes a set of pairs $[\![D]\!](v)$: for $v = \bot$ this set is the empty set, while for all $v \neq \bot$ this set is defined by all the paths from v to $\bot$, i.e., $[\![D]\!](v) = \{[w_\rho : g_\rho] \mid \rho \text{ is a path from } v \text{ to } \bot\}$. By these two assumptions, there is a correspondence between the words in $[\![D]\!](v)$ and the paths from v to $\bot$. For instance, the strings associated with n_0 in the string-DAG depicted in Figure 1a are ad with priority $0+3=3$, abc with priority $0+1+2=3$, aec with priority $0+4+2=6$ and ϵ with priority 5.

This structure is useful to store a big number of strings in a compressed manner. Further, since ϵ can only appear at the last edge of a path, by doing a DFS it can be used to retrieve all of them without repetitions and taking time linear in the length of each string. However, one can see that it is not very useful when we want to enumerate them by rank order. This motivates the following string-DAG construction. We define a function prioritize(D) that receives a string-DAG $D = (V, E)$ and returns a string-DAG $D' = (V, E')$ where each edge (u, a, g, v) of E is replaced by an edge $(u, a, g \oplus g', v)$ in E', where g' is the minimum priority in $[\![D]\!](v)$. For instance, Figure 1b shows the string-DAG resulting after applying prioritize to D of Figure 1a. Having prioritize(D) makes finding the string with minimum priority of a vertex much easier: we simply need to follow recursively the edge with minimum priority. In n_0 of Figure 1b we make the path $n_0 \xrightarrow{a,3} n_1 \xrightarrow{b,3} n_2 \xrightarrow{c,2} \bot$ and compute the minimum pair $[abc, 3]$ (the priority is retrieved from the first edge).

Before presenting the HoW implementation, we need to introduce another fully-persistent data structure. This structure is based on the Brodal queue [4], a known worst-case efficient priority queue, which we extend with the new function increaseBy. Formally, an *incremental Brodal queue*, or just a *queue*, is a fully-persistent data structure Q which stores a set $P = \{[\mathcal{E}_1 : g_1] \dots [\mathcal{E}_k : g_k]\}$, where each $\mathcal{E}_i$ is a stored element and g_i is its priority. As an abuse of notation, we often write $Q = P$. The functions to manage incremental Brodal queues include all functions for HoW except ExtendBy, namely findMin, deleteMin, add, increaseBy and meld; their definition also remains the same as for HoW. Note that we use different fonts to distinguish the operations over HoWs versus the operations over incremental Brodal queues. For example, we write FindMin for HoWs and findMin for queues. Further, this queue has two additional functions: isEmpty, that checks if the queue is the empty queue $\emptyset$; and minPrio, that returns the value g of the minimal priority among all the priorities stored. For the rest of this section we assume the existence of an incremental Brodal queue structure such that all functions run in time $\mathcal{O}(1)$ except for deleteMin, which runs in $\mathcal{O}(\log(n))$, where

Algorithm 2 HoW's implementation of Add, ExtendBy, FindMin and DeleteMin.

```
1:  procedure Add(⟨Q⟩, [a : g])
2:      return ⟨add(Q, [(a, ⟨∅⟩) : g])⟩
3:  procedure ExtendBy(⟨Q⟩, a)
4:      if isEmpty(Q) then
5:          return ⟨∅⟩
6:      return ⟨add(∅, [(a, ⟨Q⟩) : minPrio(Q)])⟩
7:  procedure FindMin(⟨Q⟩)
8:      (a, ⟨Q′⟩) ← findMin(Q)
9:      if isEmpty(Q′) then
10:         return a
11:     return FindMin(⟨Q′⟩) · a
12: procedure DeleteMin(⟨Q⟩)
13:     if isEmpty(Q) then
14:         return ⟨∅⟩
15:     (a, ⟨R⟩) ← findMin(Q)
16:     Q′ ← deleteMin(Q)
17:     ⟨R′⟩ ← DeleteMin(⟨R⟩)
18:     if isEmpty(R′) then
19:         return ⟨Q′⟩
20:     δ ← minPrio(R′) ⊕ (minPrio(R))^{-1}
21:     g ← minPrio(Q) ⊕ δ
22:     return ⟨add(Q′, [(a, ⟨R′⟩) : g])⟩
```

n is the number of pairs stored in the queue. Finally, all these operations are fully-persistent. The in-detail explanation of this structure is derived to the next section.

With the previous intuition and the structure above, we can now present the implementation for Heap of Words. A HoW h is implemented as an incremental Brodal queue Q that stores a set $\{[(a_1, h_1) : g_1], \ldots, [(a_k, h_k) : g_k]\}$, where each $a_i \in \Sigma \cup \{\epsilon\}$, each h_i is a HoW and each $g_k \in \mathbb{G}$. We write $h = \langle Q \rangle$ to make clear that we are talking about a HoW and not the queue. The empty HoW is simply the empty queue $\langle \emptyset \rangle$. Intuitively, the recursive references to HoWs are used to encode a string-DAG D; more specifically, we use it to encode prioritize$(D) = (V, E)$ and store the edges using the queue structure. For every $u \in V$, we define a HoW $h_u = \langle Q \rangle$ such that each pair $[(a, h_v) : g]$ stored in Q represents an edge $(u, a, g, v) \in E$. For instance, continuing with the example of Figure 1b, we have a HoW for each vertex: $h_\perp = \langle \emptyset \rangle$, $h_{n_2} = \langle \{[(c, h_\perp) : 2]\} \rangle$, $h_{n_1} = \langle \{[(e, h_{n_2}) : 6], [(b, h_{n_2}) : 3], [(d, h_\perp) : 3]\} \rangle$ and $h_{n_0} = \langle \{[(a, h_{n_1}) : 3], [(\epsilon, h_\perp) : 5]\} \rangle$.

We now explain the implementation of the functions defined in Section 5 to manage HoW. Consider a HoW $h = \langle Q \rangle$. For each $\text{OP} \in \{\text{Meld}, \text{IncreaseBy}\}$, the function is just applied directly to the queue, i.e., $\text{OP}(\langle Q \rangle) = \langle \text{op}(Q) \rangle$. The implementation of Add and the other functions is now described and presented in Algorithm 2.

In the case of Add(h, a), an edge is added that points to $\langle \emptyset \rangle$; this can be extended to add a word w instead by allowing that edges keep words instead of single letters. To implement ExtendBy$(\langle Q \rangle, a)$, we simply need to create a new queue containing the element $[(a, \langle Q \rangle) : \text{minPrio}(Q)]$. Note that the appended letter is added not at the end, but at the beginning, meaning that the strings are actually being stored in inverted order. This is managed in FindMin$(\langle Q \rangle)$, where the output string is inverted back. For FindMin$(\langle Q \rangle)$, to get the minimum element we recursively use findMin(Q) to find the outgoing edge with minimum priority, as we explained when the prioritize function was introduced. For DeleteMin, in order to delete the string with minimum priority, we use the fact that the set of all paths, minus the one with minimal priority, is composed by: (1) all the paths that do not start with the minimal edge, and (2) all the paths starting with the minimal edge that are followed by any path minus the one with minimal priority. For instance, in Figure 1b, the minimal path from n_0 is $\pi = n_0 \xrightarrow{a} n_1 \xrightarrow{d} \perp$. Then, the set of paths minus π is composed by (1) $n_0 \xrightarrow{\epsilon} \perp$, and (2) $n_0 \xrightarrow{a} n_1 \xrightarrow{e} n_2 \xrightarrow{c} \perp$, $n_0 \xrightarrow{a} n_1 \xrightarrow{b} n_2 \xrightarrow{c} \perp$. In procedure DeleteMin, $\langle Q' \rangle$ stores the paths of (1), while $\langle R' \rangle$ stores the paths from (2) minus the first edge (lines 16-17).

Further, since the minimal path was removed, a new priority needs to be computed for this edge, which is computed and stored as g (line 20-21). This priority is used to create an edge to R', i.e., $[(a, \langle R' \rangle) : g]$, which together represent the paths of (2). This is connected with the paths of (1), i.e., $\langle Q' \rangle$, and the result is returned in line 22. The border case case where (2) is empty is managed by lines 18-19, in which case it simply returns $\langle Q' \rangle$.

It is straightforward to check that this data structure achieves the time and space bounds given in Section 5. We end this section by arguing that the implementation of HoW is fully-persistent. For this, note that the performance of HoW relies on the implementation of incremental Brodal queues. Indeed, given that these queues are fully-persistent and each method in Algorithm 2 creates new queues without modifying the previous ones, the whole data structure is fully-persistent. Therefore, it is left to prove that we can extend Brodal queues as we already mentioned. We will show this in the last section.

7 Incremental brodal queues

In this section, we discuss how to implement an incremental Brodal queue, the last ingredient of our ranked enumeration algorithms for MSO cost functions. This data structure extends Brodal queues [4] by including the increaseBy procedure. Indeed, our construction of incremental Brodal queues follows the same approach as in [4]. We start by defining what we call an *incremental binomial heap*, for which most operations take logarithmic time, to then show how to extend it to lower the cost to constant time, except for deleteMin that takes logarithmic time. The relevant aspects for this extension (i.e., to support increaseBy) appear in the definition of the incremental binomial heap. For this reason, in this section we present only the implementation of the incremental binomial heap. The details of how to extend it to an incremental Brodal queue can be found in [4]. We start by introducing some notation.

A *multitree structure* is a pair $M = (V, \mathsf{first}, \mathsf{next}, v^0)$ where V is a set of nodes, $\mathsf{first} : V \to V \cup \{\bot\}$ and $\mathsf{next} : V \to V \cup \{\bot\}$ are functions such that $\bot \notin V$ and $v^0 \in V$ is a special node. Further, we assume that the directed graph $G_M = (V, \{(u, v) \mid \mathsf{first}(u) = v \text{ or } \mathsf{next}(u) = v\})$ is a multitree, namely, it is a directed acyclic graph (DAG) in which the set of vertices reachable from any vertex induces a tree. Let V_{v^0} denotes the reachable nodes from v^0 and $G_{v^0} = (V_{v^0}, \{(u, v) \mid \mathsf{first}(u) = v \text{ or } \mathsf{next}(u) = v\})$ the graph induced by V_{v^0}, which is a tree by definition. Note that G_{v^0} is using the first-child next-sibling encoding to form an ordered forest. To see this, let $\mathsf{next}^*(v)$ be the smallest subset of V such that $v \in \mathsf{next}^*(v)$ and $\mathsf{next}(u) \in \mathsf{next}^*(v)$ whenever $u \in \mathsf{next}^*(v)$. Then the set $\mathsf{roots} = \mathsf{next}^*(v^0)$ represents the roots of the forest and for each $v \in V_{v^0}$ the set $\mathsf{children}(v) = \mathsf{next}^*(\mathsf{first}(v))$ are the children of the node v in the forest where $\mathsf{children}(v) = \emptyset$ when $\mathsf{first}(v) = \bot$. Here both sets are ordered by the next function, then we will usually write $\mathsf{roots} = v_1, \ldots, v_j$ or $\mathsf{children}(v) = u_1, \ldots, u_k$ to denote both the elements of the set and its order. Also, we write $\mathsf{parent}(v) = u$ if $v \in \mathsf{children}(u)$ and we say that v is a leaf if $\mathsf{first}(v) = \bot$. Note that in M a node could have different "parents" (i.e., G_M is a DAG) depending on the node v^0 that we start. We say that M *forms a tree* if $\mathsf{next}(v^0) = \bot$. Furthermore, for $v \in V$ we denote by M_v the tree hanging from v, namely, M_v is equal to M with the exception that $v^0_{M_v} = v$ and $\mathsf{next}_{M_v}(v) = \bot$. As it will clear below, this encoding will be helpful to build the data structure and assure the persistent requirement.

A binomial tree of rank k is recursively defined as follows. A binomial tree of rank 0 is a leaf and a binomial tree of rank $k+1$ is a multitree structure M that forms a tree such that $\mathsf{children}(v^0) = u_k, \ldots, u_0$ and M_{u_i} is a binomial tree of rank i. If M is a binomial tree we denote its rank by $\mathrm{rank}(M)$. One can easily show by induction over the rank (see [5]) that for

every binomial tree M of ranked k, it holds that $|V_{v^0}| = 2^k$ and, thus, the number of children of each node is of logarithmic size with respect to the size of T, i.e., $|\mathsf{children}(v)| \leq \log(|V_{v^0}|)$ for every $v \in V_{v^0}$. We use this property several times throughout this section.

Fix an ordered group $(\mathbb{G}, \oplus, \mathbb{0}, \preceq)$. An incremental binomial heap over $\mathbb{G}$ is defined as a pair $H = (V, \mathsf{first}, \mathsf{next}, v^0, \Delta, \mathsf{elem}, \delta^0)$ where $(V, \mathsf{first}, \mathsf{next}, v^0)$ is a multitree structure, $\Delta : V \to \mathbb{G}$ is the delta-priority function, $\mathsf{elem} : V \to \mathcal{E}$ is the element function where $\mathcal{E}$ is the set of elements that are stored, and $\delta^0 \in \mathbb{G}$ is an initial delta value. Further, if M is the multitree structure defined by $(V, \mathsf{first}, \mathsf{next}, v^0)$ and $\mathsf{roots} = v_1, \ldots, v_n$ are its roots, then each M_{v_i} is a binomial tree with $\mathrm{rank}(M_{v_i}) < \mathrm{rank}(M_{v_{i+1}})$ for each $i < n$. In other words, an incremental binomial heap has the same underlying structure than a standard binomial heap [5]. Usually in the literature [4], a binomial heap is imposed a min-heap property, meaning that a node always has lower priority than its children, which is crucial for dequeuing elements in order. Instead, we give to our heap a different semantics by keeping the difference between nodes with the Δ-function and computing the real priority function $\mathsf{pr}_{v^0} : V_{v^0} \to \mathbb{G}$ as follows: $\mathsf{pr}_{v^0}(v) := \delta^0 \oplus \Delta(v)$ whenever v is a root of the underlying multitree structure, and $\mathsf{pr}_{v^0}(v) := \mathsf{pr}_{v^0}(u) \oplus \Delta(v)$ whenever $\mathsf{parent}(v) = u$. Given that $\mathsf{parent}(v)$ depends on the starting node v^0, then pr_{v^0} also depends on v^0. In addition, we assume that a min-heap property is satisfied over the real priority function, namely, $\mathsf{pr}_{v^0}(u) \preceq \mathsf{pr}_{v^0}(v)$ whenever $\mathsf{parent}(v) = u$. Then H is a heap where each node $v \in V$ keeps a pair $(\mathsf{elem}(v), \mathsf{pr}_{v^0}(v))$ where $\mathsf{elem}(v)$ is the stored element and $\mathsf{pr}_{v^0}(v)$ its priority in the heap. This principle of storing the deltas between nodes instead of the real priority is crucial for supporting the increased-by operation of the data structure.

Next, we show how to implement the operations of an incremental Brodal queue stated in Section 6, namely, isEmpty, increaseBy, findMin minPrio, deleteMin, add, and meld. We implement this with an incremental binomial heap where the only difference is that isEmpty and increaseBy will take constant time, and findMin minPrio, deleteMin, add, and meld will take logarithmic time. To extend incremental binomial heaps to lower the complexity of findMin minPrio, deleteMin, and add to constant time, one can use the same techniques as in [4]. Most operations of incremental binomial heaps are similar to the operations on binomial heaps (see [5]), however, for the sake of completeness we explain each one in detail, highlighting the main differences to manage the delta priorities.

From now on, fix an incremental binomial heap $H = (V, \mathsf{first}, \mathsf{next}, v^0, \Delta, \mathsf{elem}, \delta^0)$. Given that all operations must be persistent, we will usually create a copy H' of H by extending H with new fresh nodes. More precisely, we will say that H' is an *extension* of H (denoted by $H \subseteq H'$) iff $V_H \subseteq V_{H'}$ and $\mathsf{op}_{H'}(v) = \mathsf{op}_H(v)$ for every $v \in V_H$ and $\mathsf{op} \in \{\mathsf{first}, \mathsf{next}, \Delta, \mathsf{elem}\}$ (note that v^0 and δ^0 may change). Furthermore, for $H \subseteq H'$ we will say that a node $v' \in V_{H'} \setminus V_H$ is a *fresh copy* of $v \in V_H$ if v' in H' has the same structure as v in H where only the differences are defined explicitly, namely, we omit the functions that are the same as for v. For example, if we say that "v' is a fresh copy of v such that $\mathsf{next}_{H'}(v') := \perp$", this means that $\mathsf{next}_{H'}(v') := \perp$ and $\mathsf{op}_{H'}(v') = \mathsf{op}_H(v)$ for every $\mathsf{op} \neq \mathsf{next}$.

The first operation, $\mathsf{isEmpty}(H)$, can easily be implemented in constant time, by just checking whether $v^0 = \perp$ or not. Similarly, $\mathsf{increaseBy}(H, \delta)$ can be implemented in constant time by just updating δ^0 to $\delta^0 \oplus \delta$, which is the purpose of having δ^0. For $\mathsf{findMin}(H)$ or $\mathsf{minPrio}(H)$, a bit more of work is needed. Recall that a k-rank binomial tree with $|V|$ nodes satisfies $|V| = 2^k$. Given that $\mathsf{roots} = v_1, \ldots, v_n$ is a sequence of binomial trees ordered by rank, one can easily see that $n \in \mathcal{O}(\log(|V_{v^0}|)$. Therefore, we need at most a logarithmic number of steps to find the node v_i with the minimum priority and return $\mathsf{elem}(v_i)$ or $\mathsf{pr}_{v^0}(v_i)$ whenever $\mathsf{findMin}(H)$ or $\mathsf{minPrio}(H)$ is asked, respectively.

For $\mathsf{add}(H, e, g)$ or $\mathsf{deleteMin}(H)$, we reduce them to melding two heaps. For the first operation, we create a heap H' whose multitree structure has one node, call it v, $\Delta_{H'}(v) := g$, $\mathsf{elem}_{H'}(v) := e$, and $\delta^0_{H'} := \mathbb{0}$. Then we apply $\mathsf{meld}(H, H')$ obtaining a heap where the new node (e, g) is added to H. For the second operation, we remove the minimum element by creating two heaps and then apply the meld operation. Specifically, let $\mathsf{roots} = v_1, \ldots, v_n$ be the roots of H and v_i be the root with the minimum priority. Then we build two heaps H_1 and H_2 such that $H \subseteq H_i$ for $i \in \{1, 2\}$. For H_1, we extend H by creating fresh copies of all v_j, $j \neq i$. Formally, define $V_{H_1} = V_H \cup \{v'_1, \ldots, v'_n\}$ where each v'_j is a fresh copy of v_j with the exception of v'_{i-1} that we set $\mathsf{next}_{H_1}(v'_{i-1}) := v'_{i+1}$. Finally, define $v^0_{H_1} = v'_1$ as the starting node of H_1. Now, for H_2 we extend H by creating a copy of the children of v_i in H in reverse order and updating δ^0_H to $\delta^0_H \oplus \Delta_H(v_i)$ (recall that the children of a binomial tree are ordered by decreasing rank). Formally, if $\mathsf{children}_H(v_i) = u_1, \ldots, u_k$, then $V_{H_2} = V_H \cup \{u'_1, \ldots, u'_k\}$ where each u'_j is a fresh copy of u_j such that $\mathsf{next}_{H_2}(u'_j) := u'_{j-1}$ for $j > 1$ and $\mathsf{next}_{H_2}(u'_1) := \perp$. Finally, define $v^0_{H_2} := u'_k$ and $\delta^0_{H_2} := \delta^0_H \oplus \Delta_H(v_i)$. The reader can check that H_1 and H_2 are valid incremental binomial heaps and, furthermore, H_1 is H without v_i and H_2 contains only the children of v_i in reverse order. Therefore, to compute $\mathsf{deleteMin}(H)$ we return $\mathsf{meld}(H_1, H_2)$. Given that the construction of H_1 and H_2 takes at most logarithmic time in the size of H (i.e., there is at most a log number of roots or children), then the procedure takes logarithmic time. Furthermore, H was never touched and then the operation is fully-persistent.

For $\mathsf{meld}(H_1, H_2)$, we use the same algorithm as for melding two binomial heaps with two modifications that are presented here. For melding two binomial heaps, we point the reader to [5] in which this operation is well explained. For the first change, we need to update the link operation [5] of two binomial trees to support the use of the delta priorities. Given a incremental binomial heap H and its underlying multitree structure M, let v_1 and v_2 be two nodes in H such that $\Delta(v_1) \preceq \Delta(v_2)$ and M_{v_1} and M_{v_2} has the same rank k. Then the *link* of v_1 and v_2, denoted by $\mathtt{link}(H, v_1, v_2)$, outputs a pair (H', v'_1) such that H' is an extension of H and $M'_{v'_1}$ is a binomial tree of rank $k+1$ containing the nodes of M_{v_1} and M_{v_2}. Formally, $V_{H'} := V_H \cup \{v'_1, v'_2\}$ and v'_1 and v'_2 are fresh copies of v_1 and v_2 such that $\mathsf{first}_{H'}(v'_1) := v'_2$, $\mathsf{next}_{H'}(v'_2) := \mathsf{first}_H(v_1)$ and $\Delta_{H'}(v'_2) := \Delta_H(v_1)^{-1} \oplus \Delta_H(v_2)$. Note that the new node v'_1 defines a binomial tree $M'_{v'_1}$ of rank $k+1$ containing all nodes of M_{v_1} and M_{v_2}, maintaining the priorities of H and such that $\mathsf{pr}_{H'}(u) \preceq \mathsf{pr}_{H'}(u')$ whenever $u = \mathsf{parent}(u')$. The second change of the algorithm in [5] is that, before melding H_1 and H_2, we push each initial delta value to the roots of the corresponding data structures. For this, given an incremental binomial heap H we construct $H^{\downarrow}$ with $H \subseteq H^{\downarrow}$ as follows. Let $\mathsf{roots}_H = v_1, \ldots, v_k$. Then $V_{H^{\downarrow}} = V_H \cup \{v'_1, \ldots, v'_k\}$ where $v'_1, \ldots, v'_k$ are fresh copies of $v_1, \ldots, v_k$ and $\Delta_{H^{\downarrow}}(v'_i) := \delta^0_H \oplus \Delta_H(v_i)$. Furthermore, we define $v^0_{H^{\downarrow}} := v'_1$ and $\delta^0_{H^{\downarrow}} := \mathbb{0}$. Note that in $H^{\downarrow}$ we can forget about the initial delta value given that this is included in the root of each binomial tree. Finally, to meld H_1 and H_2 we construct $H_1^{\downarrow}$ and $H_2^{\downarrow}$ and then apply the melding algorithm of [5] with the updated version of the link function, $\mathtt{link}(H, v_1, v_2)$. Overall, the operation takes logarithmic time to build $H_1^{\downarrow}$ and $H_2^{\downarrow}$, and logarithmic time to meld both heaps. Moreover, given that $\mathtt{link}(H, v_1, v_2)$ and the construction of $H_1^{\downarrow}$ and $H_2^{\downarrow}$ do not modify the initial heap H, then the meld operation is persistent as well.

To finish this section, we recall that the next step is to extend the incremental binomial heap to an incremental Brodal queue. For this, we follow the same approach as [4] to lower the time complexity of find-min, add, and meld operation from logarithmic to constant time.

8 Conclusions

This paper presented an algorithm to enumerate the answers of queries over words, in an order defined by a cost function, that has a linear preprocessing and a logarithmic delay in the size of the words. We first introduced the notion of MSO cost functions, to then present a ranked enumeration scheme. This scheme relies on a particular data structure called HoW. The complexity of our algorithms depends mainly on the performance of the operations of HoW. To implement them, we extend a well known persistent data structure called Brodal queue. Thanks to this data structure, we obtain the bounds of our algorithm.

For future work, we would like to find a lower bound that justifies the logarithmic delay or whether one can achieve a better delay. We also plan to study how the introduced data structures and algorithms could be used in other enumeration schemes (e.g., relational databases). Finally, we would also like to validate our approach in practical settings.

References

1 Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

2 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, pages 22:1–22:19, 2019.

3 Guillaume Bagan. Mso queries on tree decomposable structures are computable with linear delay. In *International Workshop on Computer Science Logic*, pages 167–181. Springer, 2006.

4 Gerth Stølting Brodal and Chris Okasaki. Optimal purely functional priority queues. *Journal of Functional Programming*, 6(6):839–857, 1996.

5 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

6 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.

7 Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. *CoRR*, abs/1902.02698, 2019.

8 Johannes Doleschal, Noa Bratman, Benny Kimelfeld, and Wim Martens. The complexity of aggregates over extractions by regular expressions. In *ICDT*, 2021.

9 Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. In *ICDT*, volume 155, pages 8:1–8:18, 2020.

10 James R Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. In *STOC*, pages 109–121, 1986.

11 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. In *ICALP*, volume 3580, pages 513–525, 2005.

12 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

13 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12:1–12:51, 2015.

14 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Efficient enumeration algorithms for regular document spanners. *ACM Trans. Database Syst.*, 45(1):3:1–3:42, 2020.

15 Dominik D Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *Proceedings of PODS*, pages 137–149, 2018.

16 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Log.*, 130(1-3):3–31, 2004.

17 Jonathan S Golan. *Semirings and their Applications*. Springer Science & Business Media, 2013.

18 Alejandro Grez, Cristian Riveros, and Martín Ugarte. A Formal Framework for Complex Event Processing. In *ICDT*, pages 5:1–5:18, 2019.

19 Stephan Kreutzer and Cristian Riveros. Quantitative monadic second-order logic. In *LICS*, pages 113–122, 2013.

20 Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

21 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *Proceedings of PODS*, pages 125–136. ACM, 2018.

22 Klaus Reinhardt. The complexity of translating logic to finite automata. In *Automata logics, and infinite games*, pages 231–238. Springer, 2002.

23 Luc Segoufin. Enumerating with constant delay the answers to a query. In *Proceedings of the 16th International Conference on Database Theory*, pages 10–20, 2013.

24 Nikolaos Tziavelis, Deepak Ajwani, Wolfgang Gatterbauer, Mirek Riedewald, and Xiaofeng Yang. Optimal algorithms for ranked enumeration of answers to full conjunctive queries. *VLDB*, 13(9):1582–1597, 2020.

25 Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. Optimal join algorithms meet top-k. In *SIGMOD*, pages 2659–2665. ACM, 2020.

Approximate Similarity Search Under Edit Distance Using Locality-Sensitive Hashing

Samuel McCauley
Williams College, Williamstown, MA, USA

Abstract

Edit distance similarity search, also called approximate pattern matching, is a fundamental problem with widespread database applications. The goal of the problem is to preprocess n strings of length d, to quickly answer queries q of the form: if there is a database string within edit distance r of q, return a database string within edit distance cr of q.

Previous approaches to this problem either rely on very large (superconstant) approximation ratios c, or very small search radii r. Outside of a narrow parameter range, these solutions are not competitive with trivially searching through all n strings.

In this work we give a simple and easy-to-implement hash function that can quickly answer queries for a wide range of parameters. Specifically, our strategy can answer queries in time $\widetilde{O}(d3^r n^{1/c})$. The best known practical results require $c \gg r$ to achieve any correctness guarantee; meanwhile, the best known theoretical results are very involved and difficult to implement, and require query time that can be loosely bounded below by 24^r. Our results significantly broaden the range of parameters for which there exist nontrivial theoretical bounds, while retaining the practicality of a locality-sensitive hash function.

2012 ACM Subject Classification Information systems → Nearest-neighbor search; Theory of computation → Pattern matching

Keywords and phrases edit distance, approximate pattern matching, approximate nearest neighbor, similarity search, locality-sensitive hashing

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.21

Funding This research was supported in part by the European ResearchCouncil under the European Union's 7th Framework Programme (FP7/2007–2013) / ERC grant agreement no. 61433, by the VILLUM Foundation grant 16582, and by a Zuckerman STEM Postdoctoral Fellowship.

Acknowledgements I would like to thank Thomas Ahle, Martin Aumüller, Tobias Christiani, Tsvi Kopelowitz, Rasmus Pagh, Ely Porat, Francesco Silvestri, and Shikha Singh for many helpful comments and discussions.

1 Introduction

For a large database of items, a *similarity search* query asks which database item is most similar to the query. This leads to a basic algorithmic question: how can we preprocess the database to answer these queries as quickly as possible?

Similarity search is used frequently in a wide variety of applications. Unfortunately, for databases containing high-dimensional items, algorithm designers have had trouble obtaining bounds that are significantly faster than a linear scan of the entire database. This has often been referred to as the "curse of dimensionality." Recent work in fine-grained complexity has begun to explain this difficulty: achieving significantly better than linear search time would contradict the strong exponential time hypothesis [2, 14, 35].

However, these queries can be relaxed to *approximate similarity search* queries. For an approximation factor c, we want to find a database item that is at most a c factor less similar than the most similar item.

Approximate similarity search is fairly well-understood for many metrics; see [3] for a survey. For example, in Euclidean space we have theoretical upper bounds [4, 12], fast implementations [5, 17, 21, 25, 27, 37], and lower bounds for a broad class of algorithms [4]. Many of these results are based on *locality-sensitive hashing* (LSH), originally described in [20]. A hash is locality-sensitive if similar items are likely to share the same hash value.

When a database contains text items, a natural notion of similarity is edit distance: how many character inserts, deletes, and replacements are required to get from the query string to a database string? In fact, edit distance similarity search is frequently used in computational biology [22,24,33], spellcheckers [7,38], computer security (in the context of finding similarity to weak passwords) [28], and many more applications; see e.g. [6].

Surprisingly, finding an efficient algorithm for approximate similarity search under edit distance remains essentially open. Known results focus on methods for exact similarity search (with $c = 1$), which incur expensive query times, and on embeddings, which require very large – in fact superconstant – approximation factors c.

However, recent work provides a potential exception to this. The CGK embedding [8] is simple and practical, and embeds into Hamming space with stretch $O(r)$ – in particular, it does well when the distance between the closest strings is fairly small. EmbedJoin, a recent implementation by Zhang and Zhang [39], showed that the CGK embedding performs very well in practice. EmbedJoin first embeds each string into Hamming space using the CGK embedding. Then, the remaining nearest neighbor search[1] is done using the classic bit sampling LSH for Hamming distance. Each of these steps – both the CGK embedding and the bit sampling LSH – is repeated several times independently. This method gave orders of magnitude better performance than previous methods. Furthermore, their results greatly outperformed the worst-case CGK analysis.

Thus, several questions about using CGK for edit distance similarity search remained. Zhang and Zhang used several CGK embeddings, performing a sequence of Hamming distance hashes for each – can these two steps be combined into a single method to improve performance? Meanwhile, their tests focused on practical datasets; is it possible to provide worst-case bounds for this method, ensuring good performance for any dataset?

In this paper we answer these questions in the affirmative. In doing so, we give the first locality-sensitive hash for edit distance with worst-case guarantees.

1.1 Results

The main result of our paper is the first locality-sensitive hash for edit distance. We analyze the performance of this hash when applied to the problems of approximate similarity search and approximate nearest neighbor search, obtaining time bounds that improve on the previously best-known bounds for a wide range of important parameter settings.

Let n be the number of strings stored in the database. We assume that all query strings and all database strings have length at most d. We assume $d = O(n)$ and the alphabet size is $O(n)$.[2]

[1] Zhang and Zhang investigated similarity *joins*, in which all similar pairs in a set are returned, rather than preprocessing for individual nearest neighbor queries. However, their ideas can be immediately generalized.

[2] Usually d and the alphabet size are much smaller. If this assumption does not hold, it is likely that a completely different approach will be more successful: for example, if $d = \text{poly}(n)$, then the method used to calculate the edit distance between two strings becomes critically important to the query time.

Our first result analyzes the time and space required by our LSH to solve the approximate similarity search problem. This data structure works for a fixed radius r: for each query, if there exists a database point within distance r, we aim to (with good probability) return a database point within distance cr. We use $\widetilde{O}(f(n))$ as a shorthand for $O(f(n) \cdot \text{polylog} f(n))$.

▶ **Theorem 1.** *There exists a data structure answering Approximate Similarity Search queries under Edit Distance in $\widetilde{O}(d3^r n^{1/c})$ time per query, and $\widetilde{O}(d3^r n^{1+1/c})$ preprocessing time and space.*

We also give a data structure that answers queries where the distance r to the closest neighbor is not known during preprocessing. We call this the approximate *nearest* neighbor search problem.

▶ **Theorem 2.** *There exists a data structure answering Approximate Nearest Neighbor Search queries under Edit Distance in $\widetilde{O}(d3^r n^{1/c})$ time per query and $\widetilde{O}(dn^2)$ preprocessing time and space.*

Implications for Related Problems. Our results lead to immediate bounds for similarity join, where all close pairs in a database are computed; see e.g. [34, 39, 40].

Much of the previous work on approximate similarity search under edit distance considered a variant of this problem: there is a long text T, and we want to find all locations in T that have low edit distance to the query q. Our results immediately apply to this problem by treating all d-length substrings of T as the database of items.

Frequently, practical situations may require that we find all of the neighbors with distance at most r, or (similarly) the k closest neighbors. See e.g. [1] for a discussion of this problem in the context of LSH. Our analysis immediately applies to these problems. However, if there are k desired points, the running times given in Theorems 1 and 2 increase by a factor k.

1.2 Comparison to Known Results

In this section, we give a short summary of some key results for edit distance similarity search. We focus on algorithms that have worst-case query time guarantees. We refer the reader to [39, 40] as good resources for related practical results, and [6, 26, 31] for a more extensive discussion of related work on the exact problem (with $c = 1$).

Exact Similarity Search Under Edit Distance. Exact similarity search under edit distance (i.e. with $c = 1$) has been studied for many years. We focus on a breakthrough paper of Cole, Gottlieb, and Lewenstein that achieved space $O(n5^r(1.5r + \log n)^r/r!)$ and query time $O(d + 6^r(1.5r + \log n)^r/r!)$ [15].[3] We will call this structure the CGL tree. These bounds stand in contrast to previous work, which generally had to assume that the length of the strings d or the size of the alphabet $|\Sigma|$ was a constant to achieve similar bounds. Later work has improved on this result to give similar query time with linear space [9].

Before comparing to our bounds, let us lower bound the CGL tree query time – while this gives a lower bound on an upper bound (an uncomfortable position since we are not specifying its exact relationship to the data structure), it will be helpful to get a high-level idea of how the performance of the CGL tree compares to our results. Using Sterling's approximation, and dropping the $+d$ term, we can simplify the query time to

[3] These bounds are a slight simplification of the actual results using the AM-GM inequality.

$\widetilde{O}((6e/r)^r(1.5r+\log n)^r) \leq \widetilde{O}((9e)^r(1+(\log n)/(1.5r))^r)$. From this final equation, we can see that even for very small n, the guaranteed query time is at least $(9e)^r > 24^r$; if $\log n \gg 1.5r$ it can become much worse.

Comparing the $(9e)^r$ term with our query time of $\widetilde{O}(d3^r n^{1/c})$, it seems that which is better depends highly on the use case – after all, we're exchanging a drastically improved exponential term in r for a polynomial term in n.

However, there is reason to believe that our approach has some significant advantages. First, for c bounded away from 1, with moderate n and small d, the CGL query time rapidly outpaces our own even for small r. Let's do a back-of-the-envelope calculation with some reasonable parameters – we ignore constants here, but note that slight perturbations in r easily make up for such discrepancies. If we have 400k strings of 500 characters[4] with $c = 1.5$, $6^r(1.5r + \log n)^r/r! \geq d3^r n^{1/c}$ for $r > 4$. In other words, even for very small search radii and fairly large n (where the CGL tree excels), the large terms in the base of r can easily overcome a polynomial-in-n term. Second, the constants in the CGL tree seem to be unfavorable: the CGL tree uses beautiful but nontrivial data structures for LCA and LCP that may add to the constants in the query time. In other words, it seems likely that the CGL tree is most viable for even smaller values of r than the above analysis would indicate.

We suspect that these complications are part of the reason why state-of-the-art practical edit distance similarity search methods are based on heuristics or embeddings, rather than tree-based methods (see e.g. [40]).

Approximate Similarity Search Under Edit Distance. Previous results for approximate similarity search with worst-case bounds used either product metrics, or embeddings into L_1.

In techniques based on *product metrics*, each point is mapped into several separate metrics. The distance between two points is defined as their maximum distance in any of these metrics. Using this concept, Indyk provided an extremely fast (but large) nearest-neighbor data structure requiring $O(d)$ query time and $O(n^{d^{1/(1+\log c)}})$ space for any $c \geq 3$ [19].

Embedding into L_1. Because there are approximate nearest neighbor data structures for L_1 space that require $n^{1/c+o(1)}$ time and $n^{1+1/c+o(1)}$ space,[5] an embedding into L_1 with stretch α leads to an approximate nearest neighbor data structure with query time $n^{\alpha/c+o(1)}$ for $c > \alpha$.

A long line of work on improving the stretch of embedding edit distance into L_1 ultimately resulted in a deterministic embedding with stretch $\exp(\sqrt{\log d}/\log\log d)$ [32].

More recently, the CGK embedding parameterized by r instead of d, giving an embedding into Hamming space[6] with stretch $O(r)$ [8]. However, the constants proven in the CGK result are not very favorable – the upper limit on overall stretch given in the paper is $2592r$ (though this may be improvable with tighter random walk analysis). Thus, using the CGK embedding, and then performing the standard bit sampling LSH for Hamming distance on the result, gives an approximate similarity search algorithm with query time $n^{2592r/c+o(1)}$ so long as $c > 2592r$. We describe in detail how our approach improves on this method in Appendix 1.3.

[4] These are the parameters of the UniRef90 dataset from the UniProt Project `http://www.uniprot.org/`, one protein genome dataset used as an edit distance similarity search benchmark [39, 40]; other genomic datasets have (broadly) similar parameters.

[5] This can be improved to $n^{1/(2c-1)+o(1)}$ and $n^{1+1/(2c-1)+o(1)}$ time and space respectively using data-dependent techniques, and can be further generalized to other time-space tradeoffs; see [4].

[6] Hamming space and L_1 have essentially the same state-of-the-art similarity search bounds.

Zhang and Zhang [39] implemented a modified and improved version of this approach. Their results far outperformed the above worst-case analysis. Closing this gap between worst-case analysis and practical performance is one contribution of this work.

There is a lower bound of $\Omega(\log d)$ for the stretch of any embedding of edit distance into L_1 [23]. This implies that embedding into L_1 is a hopeless strategy for $c < \log d$, whereas we obtain nontrivial bounds even for constant c. Thus, for this parameter range, using a locality-sensitive hash is fundamentally more powerful than embedding into L_1.

Locality-Sensitive Hashing. An independent construction of an LSH for edit distance was given by Marçais et al. [29]. Their work uses a fundamentally different approach, based on an ordered min-hash of k-mers. Their results include bounds proving that the hash is locality-sensitive; however, they do not place any worst-case guarantees on the gap between the probability that close points collide and the probability that far points collide.

Exponential search cost. To our knowledge, a trivial brute force scan is the only algorithm for approximate similarity search under edit distance whose worst-case cost is not exponential in the search radius r. While we significantly improve this exponential term, removing it altogether remains an open problem. A recent result of Cohen-Addad et al. gave lower bounds showing that, assuming SETH, there exist parameter settings such that cost exponential in r is required for any edit distance similarity search algorithm [14]. Their results do not immediately imply that the exponential-in-r term in our query time is necessary (since the $n^{1/c}$ term satisfies their lower bound for sufficiently small c); however, this may give some indication as to why removing this exponential term has proven so challenging.

1.3 Technical Overview and Comparison to the CGK Embedding

Our hash function follows the same high-level structure as the CGK embedding [8]. In fact, our hash reduces to their embedding by omitting the appended character \$, and setting $p_a = 1/2$ and $p_r = 0$ (these parameters are defined in Section 3).

However, our hash has two key differences over simply using the CGK embedding to embed into Hamming space, and then using bit sampling. These differences work together to allow us to drastically improve the $n^{2592r/c+o(1)}$ bound we obtained in Section 1.2.

First, we modify p_a; that is, we modify the probability that we stay on a single character x_i of the input string for multiple iterations. Second, we combine the embedding and bit sampling into a single step – this means that we can take the embedding into account when deciding whether to sample a given character.

Combining into one step already gives an inherent improvement. After embedding, we do not want to sample a "repeated" character – this is far less useful than sampling a character the last time it is written, after the hash has attempted to align them. Thus, we only sample a character (with probability $1 - p_r$) the last time that character is written.

However, the significant speedup comes from using *repeated* embeddings – in short, at a high level, each LSH in our approach consists of a single CGK embedding with a single bit sampling LSH. If a single embedding is used, the performance of the algorithm as a whole has the expected stretch of that single embedding as a bottleneck. As a result, the expected stretch winds up in the exponent of n, and c must be at least as large as the expected stretch to guarantee correctness. By repeatedly embedding, our bounds instead depend (in a sense) on the *best*-case stretch over the many embeddings.

These repeated embeddings is where these two differences – modifying p_a and integrating into a single LSH – act in concert. A back-of-the-envelope calculation implies that a CGK embedding will have stretch[7] 2 with probability at least $1/4^r$. This analysis seems difficult to tighten: if we perform 4^r embeddings, how well will we do in the cases that don't have $O(1)$ stretch? Meanwhile, any constant loss in the analysis winds up in the exponent of n. Overall, with the CGK embedding as a black box, a full analysis would require an analysis (with tight constants) detailing the probability that an embedding has any given stretch. Instead, by combining these approaches in a single LSH, we can instead model the entire problem as a single random walk in a two-dimensional grid.

Overall, a combined approach gives better worst-case performance, and a unified (and likely simpler) framework for analysis.

2 Model and Preliminaries

We denote the alphabet used in our problem instance as Σ. We use two special characters $\perp$ and \$, which we assume are not in Σ. The hash appends \$ to each string being hashed; we call a string \$-terminal if its last character is \$ and it does not contain another \$.

We index into strings using 0-indexed subscripts; x_0 is the first character of x and x_i is the $i+1$st character. We use $x[i]$ to denote the prefix of x of length i; thus $x[i] = x_0 \dots x_{i-1}$. Finally, we use $x \circ y$ to denote the concatenation of two strings x and y, and $|x|$ to denote the length of a string x.

2.1 Edit Distance

Edit distance is defined using three operations: inserts, deletes, and replacements. Given a string $x = x_1 x_2 \dots x_d$, inserting a character σ at position i results in a string $x' = x_1 \dots x_{i-1} \sigma x_i \dots x_d$. Replacing the character at position i with σ results in $x' = x_1 \dots x_{i-1} \sigma x_{i+1} \dots x_d$. Finally, deletion of the character at position i results in $x' = x_1 \dots x_{i-1} x_{i+1} \dots x_d$. We refer to these three operations as *edits*. The edit distance from x to y is defined as the smallest number of edits that must be applied to x to obtain y. We denote this as $\mathrm{ED}(x, y)$.

2.2 Model and Problem Definition

In this paper we solve the approximate similarity search problem under edit distance, which can be defined as follows.

▶ **Definition 3** (Approximate Similarity Search Under Edit Distance)**.** *Given a set of n strings S and constants c and r, preprocess S to quickly answer queries of the form, "if there exists a $y \in S$ with $\mathrm{ED}(q, y) \le r$, return a $y' \in S$ with $\mathrm{ED}(q, y') \le cr$ with probability $> 1/10$."*

The above is sometimes called the approximate *near* neighbor problem. The constant $1/10$ is arbitrary and can be increased to any desired constant without affecting our final bounds.

Oftentimes, we want to find the nearest database item to each query rather than parameterizing explicitly by r.

[7] To be more precise, with probability $1/4^r$ one string with distance r from the query will have embedded Hamming distance r, while all strings with distance x will have embedded Hamming distance $\ge x/2$.

▶ **Definition 4** (Approximate Nearest Neighbor Search Under Edit Distance). *Given a set of n strings S and a constant c, preprocess S to quickly answer queries of the form, "for the smallest r such that there exists a $y \in S$ with $\mathrm{ED}(q, y) \leq r$, return a $y' \in S$ with $\mathrm{ED}(q, y') \leq cr$ with probability $> 1/10$."*

For most previous LSH-based approaches, efficient Nearest Neighbor Search algorithms follow immediately from Approximate Similarity Search algorithms using the black box reduction of Har-Peled, Indyk, and Motwani [18]. However, the exponential dependence on r in our bounds requires us to instead use a problem-specific approach.

2.3 Locality-Sensitive Hashing

A hash family is *locality sensitive* if close elements are more likely to hash together than far elements. Locality-sensitive hashing is one of the most effective methods for approximate similarity search in high dimensions [4, 10, 18, 20].

▶ **Definition 5** (Locality-Sensitive Hash). *A hash family $\mathcal{H}$ is (r, cr, p_1, p_2)-sensitive for a distance function $d(x, y)$ if*

- *for all x_1, y_1 such that $d(x_1, y_1) \leq r$, $\Pr_{h \in \mathcal{H}}(h(x_1) = h(y_1)) \geq p_1$, and*
- *for all x_2, y_2 such that $d(x_2, y_2) \geq cr$, $\Pr_{h \in \mathcal{H}}(h(x_1) = h(y_1)) \leq p_2$.*

Some previous work (i.e. [11, 16]) has a stricter definition of locality sensitive hash: it requires that there exists a function f such that $\Pr(h(x) = h(y)) = f(d(x, y))$. Our hash function does not satisfy this definition; the exact value of x and y is necessary to determine their collision probability (see Lemma 16 for example).

A Note on Concatenating Hashes. Most previous approaches to nearest neighbor search begin with an LSH family that has $p_1, p_2 = \Omega(1)$. A logarithmic number of independent hashes are concatenated together so that the concatenated function has collision probability $1/n$. This technique was originally developed in [20], and has been used extensively since; e.g. in [1, 4, 13].

However, in this paper, we use a single function each time we hash. We directly set the hash parameters to achieve a desirable p_1 and p_2 (in particular, we want $p_2 \approx 1/n$). This is due to the stray constant term in Lemma 15. While our hash could work via concatenating several copies of a relatively large-probability[8] LSH, this would result in a data structure with larger space and slower running time. One interesting implication is that, unlike many previous LSH results, our running time is not best stated with a parameter $\rho = \log p_1 / \log p_2$ – rather, we choose our hashing parameters to obtain the p_1 and p_2 to give the best bounds for a given r, c, and n.

3 The Locality-Sensitive Hash

Each hash function from our family maps a string x of length d with alphabet Σ to a string $h(x)$ with alphabet $\Sigma \cup \{\perp\}$ of length $O(d + \log n)$. The function scans over x one character at a time, adding characters to $h(x)$ based on the current character of x and the current length of $h(x)$. Once the function has finished scanning x, it stops and outputs $h(x)$.

At a high level, for two strings x and y, our hash function can be viewed as randomly guessing a sequence of edits T, where $h(x) = h(y)$ if and only if applying the edits in T to x obtains y. Equivalently, one can view the hash as a random walk through the dynamic

[8] Although less than constant – Lemma 14 and the assumption that $p \leq 1/3$ implies $p_1 \leq (1/3)^r$.

programming table for edit distance, where matching edges are traversed with probability 1, and non-matching edges are traversed with a tunable probability $p \leq 1/3$. We discuss these relationships in Section 4.1.

Note the contrast with the CGK embedding, which uses a similar mechanism to guess the *alignment* between the two strings for each mismatch, rather than addressing each edit explicitly. This difference is key to our improved bounds; see Section 1.3.

Parameters of the Hash Function. We parameterize our algorithm using a parameter $p \leq 1/3$. By selecting p we can control the values of p_1 and p_2 attained by our hash (see Lemmas 14 and 15). We will describe how to choose p to optimize nearest neighbor search performance for a given r, c, and n in Section 4.3. We split p into two separate parameters p_a and p_r defined as $p_a = \sqrt{p/(1+p)}$ and $p_r = \sqrt{p}/(\sqrt{1+p} - \sqrt{p})$. Since $p \leq 1/3$, we have $p_a \leq 1/2$ and $p_r \leq 1$. For the remainder of this section, we will describe how the algorithm behaves using p_a and p_r. The rationale behind these values for p_a and p_r will become clear in the proof of Lemma 13 – in short, our choice of p_a and p_r ensures that each type of edit is guessed with the same probability.

Underlying Function. Each hash function in our hash family has an *underlying function* that maps each *(character, hash position)* pair to a pair of uniform random real numbers: $\rho : \Sigma \cup \{\$\} \times \{1, \ldots, 8d/(1-p_a) + 6\log n\} \to [0,1) \times [0,1)$.[9] We discuss how to store these functions and relax the assumption that these are real numbers in Section 4.5.

The only randomness used in our hash function is given by the underlying function.[10] In particular, this means that two hash functions h_1 and h_2 have identical outputs on all strings if their underlying functions ρ_1 and ρ_2 are identical. Thus, we pick a random function from our hash family by sampling a random underlying function. We use $h_\rho(x)$ to denote the hash of x using underlying function ρ.

The key idea behind the underlying function is that the random choices made by the hash depend only on the current character seen in the input string, and the current length of the output string. This means that if two strings are aligned – in particular, if the "current" character of x matches the "current" character of y – the hash of each will make the same random choices, so the hashes will stay the same until there is a mismatch. This is the "oblivious synchronization mechanism" used in the CGK embedding [8].

3.1 How to Hash

A hash function h is selected from the family $\mathcal{H}$ by sampling a random underlying function ρ. We denote the hash of a string x using ρ as $h_\rho(x)$. The remainder of this section describes how to determine $h_\rho(x)$ for a given x and ρ.

To hash x, the first step is to append \$ to the end of x to obtain $x \circ \$$. We will treat $x \leftarrow x \circ \$$ as the input string from now on – in other words, we assume that x is \$-terminal. Let i be the current index of x being scanned by the hash function. We will build up $h_\rho(x)$ character-by-character, storing intermediate values in a string s. The hash begins by setting $i = 0$, and s to the empty string.

[9] Adding \$ to the alphabet allows us to hash past the end of a string – this helps with edits that append characters.

[10] In fact, the underlying function is a generalization of the random string used in the CGK embedding.

Algorithm 1 Calculating $h_\rho(x)$.

```
1: i ← 0
2: Create an empty string s
3: while i < |x| and |s| < 8d/(1 − p_a) + 6 log n do
4:     (r_1, r_2) ← ρ(x_i, |s|)
5:     if r_1 ≤ p_a then
6:         Append ⊥ to s
7:     else if r_2 ≤ p_r then
8:         Append ⊥ to s
9:         i ← i + 1
10:    else
11:        Append x_i to s
12:        i ← i + 1
13: return s
```

$x = abc$
$h(x) = \perp a \perp \perp \perp \perp$
$y = bac$
$h(y) = \perp a \perp \perp \perp \perp$
$z = cba$
$h(z) = c \perp \perp a \$$

x_i \ $\lvert s\rvert$	0	1	2	3	4	5
a	$(0.1, 0.7)$	$(0.9, 0.6)$	$(0.1, 0.7)$	$(0.6, 0.8)$	$(0.2, 0.3)$	$(0.5, 0.6)$
b	$(0.6, 0.3)$	$(0.8, 0.3)$	$(0.8, 0.2)$	$(0.9, 0.4)$	$(0.1, 0.1)$	$(0.1, 0.5)$
c	$(0.7, 0.6)$	$(0.5, 0.9)$	$(0.1, 0.9)$	$(0.2, 0.8)$	$(0.7, 0.4)$	$(0.4, 0.6)$
$\$$	$(0.1, 0.4)$	$(0, 0.1)$	$(0.1, 0.3)$	$(0.8, 0.7)$	$(0.9, 0.5)$	$(0.6, 0)$

Figure 1 Example of how to hash three strings x, y, and z with underlying function ρ_1 (given in the table on the right). We use $\Sigma = \{a, b, c\}$ and $p = 1/8$, so $p_a = 1/3$ and $p_r = 1/2$. For simplicity, we round the values of ρ_1 to the first decimal place, and only give ρ_1 for $|s| \leq 5$.

The hash function repeats the following process while $i < |x|$ and[11] $|s| < 8d/(1 - p_a) + 6 \log n$. The hash first stores the current value of the underlying function based on x_i and $|s|$ by setting $(r_1, r_2) \leftarrow \rho(x_i, |s|)$. The hash performs one of three actions based on r_1 and r_2; in each case one character is appended to the string s. We name these cases a *hash-insert*, *hash-replace*, and *hash-match*.

- **If $r_1 \leq p_a$, hash-insert:** append $\perp$ to s.
- **If $r_1 > p_a$ and $r_2 \leq p_r$, hash-replace:** append $\perp$ to s and increment i.
- **If $r_1 > p_a$ and $r_2 > p_r$, hash-match:** append x_i to s and increment i.

When $i \geq |x|$ or $|s| \geq 8d/(1 - p_a) + 6 \log n$, the hash stops and returns s as $h_\rho(x)$.

We provide pseudocode for this method in Algorithm 1, and an example hash in Figure 1.

4 Analysis

In this section we show how analyze the hash given in Section 3, proving Theorems 1 and 2.

We begin in Section 4.1 with some structure that relates hash collisions between two strings x and y with sequences of edits that transform x into y. We use this to bound the probability that x and y collide in Section 4.2. With this we can prove our main results in Sections 4.3 and 4.4. Finally we discuss how to store the underlying functions in Section 4.5.

[11] The requirement $|s| < 8d/(1 - p_a) + 6 \log n$ is useful to bound the size of the underlying function in Section 4.5. We show in Lemma 6 that this constraint is very rarely violated.

4.1 Interpreting the Hash

In this section, we discuss when two strings x and y hash (with underyling function ρ) to the same string $h_\rho(x) = h_\rho(y)$.

We define three sequences to help us analyze the hash. In short, the *transcript* of x and ρ lists the decisions made by the hash function as it scans x using the underlying function ρ. The *grid walk* of x, y, and ρ is a sequence based on the transcripts (under ρ) of x and y – it consists of some edits, and some extra operations that help keep track of how the hashes of x and y interact. Finally, the *transformation* of x, y, and ρ is a sequence of edits based on the grid walk of x, y, and ρ.

Using these three sequences, we can set up the basic structure to bound the probability that x and y hash together using their edit distance. We use these definitions to analyze the probability of collision in Section 4.2.

Transcripts. A *transcript* is a sequence of hash operations: each element of the sequence is a hash-insert, hash-replace, or hash-match. Essentially, the transcript of x and ρ, denoted $\tau(x, \rho)$, is a log of the actions taken by the hash on string x using underlying function ρ.

We define an index function $i(x, k, \rho)$. The idea is that $i(x, k, \rho)$ is the value of i when the kth hash character is written when hashing x using underlying function ρ.

We set $i(x, 0, \rho) = 0$ for all x and ρ. Let $(r_{1,k}, r_{2,k}) = \rho(x_{i(x,k,\rho)}, k)$. We can now recursively define both $\tau(x, \rho)$ and $i(x, k, \rho)$. We denote the kth character of $\tau(x, \rho)$ using $\tau_k(x, \rho)$.

- If $r_{1,k} \le p_a$, then $i(x, k+1, \rho) = i(x, k, \rho)$, and $\tau_k(x, \rho) =$ hash-insert.
- If $r_{1,k} > p_a$ and $r_{2,k} \le p_r$, then $i(x, k+1, \rho) = i(x, k, \rho) + 1$, and $\tau_k(x, \rho) =$ hash-replace.
- If $r_{1,k} > p_a$ and $r_{2,k} > p_r$, then $i(x, k+1, \rho) = i(x, k, \rho) + 1$, and $\tau_k(x, \rho) =$ hash-match.

A transcript $\tau(x, \rho)$ is *complete* if $|\tau(x, \rho)| < 8d/(1 - p_a) + 6 \log n$.

▶ **Lemma 6.** *For any string x of length d, $\Pr_\rho[\tau(x, \rho)$ is complete$] \ge 1 - 1/n^2$.*

Proof. If $\tau(x, \rho)$ has ℓ hash-insert operations, then $|\tau(x, \rho)| \le d + \ell$. We bound the probability that $\ell > 7d/(1 - p_a) + 6 \log n$.

For each character in x, we can model the building of $\tau(x, \rho)$ as a series of independent coin flips. On heads (with probability p_a), ℓ increases; on tails the process stops. Thus we expect $1/(1 - p_a)$ hash-insert operations for each character of x, and at most $d/(1 - p_a)$ hash-insert operations overall.

Using standard multiplicative Chernoff bounds (i.e. [30, Exercise 4.7]), the probability that $\ell > 7d/(1 - p_a) + 6 \log n$ is at most $\exp(-(6d(1 - p_a) + 6 \log n)/3) < 1/n^2$. ◀

Grid Walks. A *grid walk* $g(x, y, \rho)$ for two strings x and y and underlying function ρ is a sequence that helps us examine how $h_\rho(x)$ and $h_\rho(y)$ interact – it is a bridge between the transcript of x, y and ρ, and the transformation induced by x, y, and ρ (which is a sequence of edits). We formally define the grid walk, and discuss how it corresponds to a random walk in a graph. This graph is closely based on the dynamic programming table for x and y.

The grid walk is a sequence of length $\max\{|\tau(x, \rho)|, |\tau(y, \rho)|\}$. The grid walk has an alphabet of size 6: each character is one of $\{\texttt{insert}, \texttt{delete}, \texttt{replace}, \texttt{loop}, \texttt{match}, \texttt{stop}\}$. At a high level, `insert`, `delete`, and `replace` correspond to string edits – for example, `insert` corresponds to the index of x being incremented while the index of y stays the same (as if we inserted the corresponding character into y). `loop` corresponds to both strings writing $\perp$

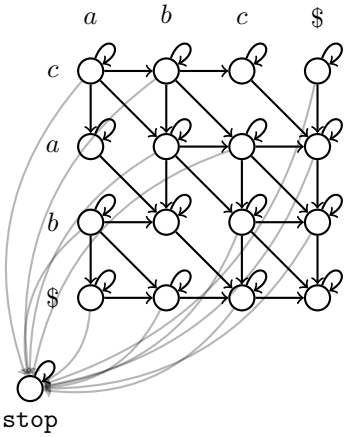


Figure 2 This figure shows $G(x, y)$ for $x = abc\$$ and $y = cab\$$. For clarity, all edge labels are ommited and `stop` edges are partially transparent.

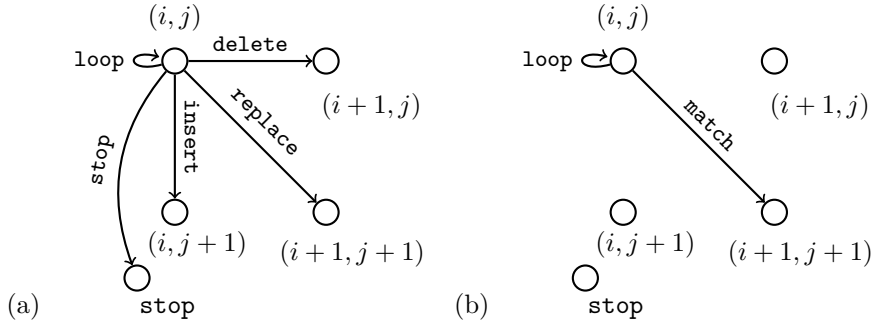


Figure 3 The edges for a single node (i, j) with $i < |x| - 1$ and $j < |y| - 1$. (a) represents the edges if $x_i \neq y_j$; (b) represents the edges if $x_i = y_j$.

without increasing i; the process "loops" and we continue with nothing changed except the length of the hash. `match` corresponds to the case when both hashes simultaneously evaluate the same character – after a sequence of `loop` operations, they will `match` by both writing out either the matching character or $\perp$ to their respective hashes. `stop` is a catch-all for all other cases: the strings write out different characters, the hashes are no longer equal, and the analysis stops.

We define a directed graph $G(x, y)$ to help explain how to construct the walk. Graph $G(x, y)$ is a directed graph with $|x||y| + 1$ nodes, corresponding roughly to the dynamic programming table between x and y. We label one node as the ***`stop`*** *node*. We label the other $|x||y|$ nodes using two-dimensional coordinates (i, j) with $0 \leq i < |x|$, and $0 \leq j < |y|$.

We now list all arcs between nodes. We label each with a grid walk character; this will be useful for analyzing $g(x, y, \rho)$. Consider an (i, j) with $0 \leq i < |x| - 1$ and $0 \leq j < |y| - 1$. For any (i, j) with $x_i \neq y_j$, we place five arcs:

- a `delete` arc from (i, j) to $(i + 1, j)$,
- a `replace` arc from (i, j) to $(i + 1, j + 1)$,
- an `insert` arc from (i, j) to $(i, j + 1)$,
- a `loop` arc from (i, j) to (i, j), and
- a `stop` arc from (i, j) to the `stop` node.

Table 1 This table defines a grid walk for non-matching characters in strings x and y, given the corresponding transcripts.

$\tau_k(x,\rho)$	$\tau_k(y,\rho)$	$g_k(x,y,\rho)$
hash-replace	hash-replace	`replace`
hash-replace	hash-insert	`delete`
hash-insert	hash-replace	`insert`
hash-insert	hash-insert	`loop`
hash-match	-	`stop`
-	hash-match	`stop`

These arcs are shown in Figure 3a. For any (i,j) with $x_i = y_j$, we place two edges: a `match` arc from (i,j) to $(i+1,j+1)$, and a `loop` arc from (i,j) to (i,j); see Figure 3b.

The rightmost and bottommost nodes of the grid are largely defined likewise, but arcs that lead to nonexistent nodes instead lead to the `stop` node.[12] For $0 \le j < |y|-1$ there is an `insert` arc from $(|x|-1,j)$ to $(|x|-1,j+1)$ a `stop` arc, `delete` arc, and `replace` arc from $(|x|-1,j)$ to the `stop` node, and a `loop` arc from $(|x|-1,j)$ to $(|x|-1,j)$. For $0 \le i < |x|-1$, there is a `delete` arc from $(i,|y|-1)$ to $(i+1,|y|-1)$, a `stop` arc, an `insert` arc, and a `replace` arc from $(i,|y|-1)$ to the `stop` node, and a `loop` arc from $(i,|y|-1)$ to $(i,|y|-1)$. Finally, node $(|x|-1,|y|-1)$ has a `loop` arc to $(|x|-1,|y|-1)$. See Figure 2.

The stop node has (for completeness) six self loops with labels `match`, `insert`, `replace`, `delete`, `loop`, and `stop`.

We now define the grid walk $g(x,y,\rho)$. We will use $G(x,y)$ to relate $g(x,y,\rho)$ to $h_\rho(x)$ and $h_\rho(y)$ in Lemmas 7 and 8.

We determine the kth character of $g(x,y,\rho)$, denoted $g_k(x,y,\rho)$, using $\tau_k(x,\rho)$ and $\tau_k(y,\rho)$, as well as $x_{i(x,k,\rho)}$ and $y_{i(y,k,\rho)}$. For $k > \min\{|\tau(x,\rho)|, |\tau(y,\rho)|\}$, $g_k(x,y,\rho) =$ `stop`.

If $x_{i(x,k,\rho)} \neq y_{i(y,k,\rho)}$, we define $g_k(x,y,\rho)$ using Table 1.

If $x_{i(x,k,\rho)} = y_{i(y,k,\rho)}$, then $\tau_k(x,\rho) = \tau_k(y,\rho)$. If $\tau_k(x,\rho) = \tau_k(y,\rho)$ is a hash-insert, then $g_k(x,y,\rho) =$ `loop`; otherwise, $g_k(x,y,\rho) =$ `match`.

We say that a grid walk is *complete* if both $\tau(x,\rho)$ and $\tau(y,\rho)$ are complete. We say that a grid walk is *alive* if it is complete and it does not contain `stop`.

The next lemma motivates this definition: the grid walk defines a path through the grid corresponding to the hashes of x and y.

▶ **Lemma 7.** *Consider a walk through $G(x,y)$ which at step i takes the edge with label corresponding to $g_i(x,y,\rho)$. Assume k is such that the prefix $g(x,y,\rho)[k]$ of length k is alive. Then after k steps, the walk arrives at node $(i(x,k,\rho), i(y,k,\rho))$.*

Proof. Our proof is by induction on k. We prove both that the walk arrives at node $(i(x,k,\rho), i(y,k,\rho))$, and that the walk is well-defined: the next character in $g(x,y,\rho)$ always corresponds to an outgoing edge of the current node.

For the base case $k = 0$ the proof is immediate, since $(i(x,0,\rho), i(y,0,\rho)) = (0,0)$. Furthermore, node $(0,0)$ has an outgoing `match` edge if and only if $x_0 = y_0$ (otherwise it has an outgoing `insert`, `delete`, and `replace` edge); similarity, $g_0(x,y,\rho) =$ `match` only if $x_0 = y_0$ (the rest of the cases follow likewise).

[12] Since x and y are \$-terminal, these nodes never satisfy $x_i = y_j$ except at $(|x|-1,|y|-1)$

Assume that after $k-1$ steps, the walk using $g(x,y,\rho)[k-1]$ arrives at node $(i(x,k-1,\rho), i(y,k-1,\rho))$. We begin by proving that the walk remains well-defined. We have $g_{k-1}(x,y,\rho) = \texttt{match}$ only if $x_{i(x,k-1,\rho)} = y_{i(y,k-1,\rho)}$; in this case $(i(x,k-1,\rho), i(y,k-1,\rho))$ has an outgoing `match` edge. We have $g_{k-1}(x,y,\rho) = \texttt{insert}$ (or `delete` or `replace`) only if $x_{i(x,k-1,\rho)} \neq y_{i(y,k-1,\rho)}$; again, node $(i(x,k-1,\rho), i(y,k-1,\rho))$ has the corresponding outgoing edge. All nodes have outgoing `loop` and `stop` edges.

Now we show that after k steps, the walk using $g(x,y,\rho)[k]$ arrives at node $(i(x,k,\rho), i(y,k,\rho))$. We split into five cases based on $g_{k-1}(x,y,\rho)$ (if $g_{k-1}(x,y,\rho) = \texttt{stop}$ the lemma no longer holds).

- `replace`: We have $\tau_k(x,\rho) =$ hash-replace, and $\tau_k(y,\rho) =$ hash-replace. Thus, $i(x,k,\rho) = i(x,k-1,\rho)+1$ and $i(y,k,\rho) = i(y,k-1,\rho)+1$. In $G(x,y)$, the edge labeled `replace` leads to node $(i(x,k-1,\rho)+1, i(y,k-1,\rho)+1)$.
- `match`: We have $\tau_k(x,\rho) =$ hash-replace, and $\tau_k(y,\rho) =$ hash-replace. Thus, $i(x,k,\rho) = i(x,k-1,\rho)+1$ and $i(y,k,\rho) = i(y,k-1,\rho)+1$. In $G(x,y)$, the edge labeled `match` leads to node $(i(x,k-1,\rho)+1, i(y,k-1,\rho)+1)$.
- `delete`: We have $\tau_k(x,\rho) =$ hash-replace, and $\tau_k(y,\rho) =$ hash-insert. Thus, $i(x,k,\rho) = i(x,k-1,\rho)+1$ and $i(y,k,\rho) = i(y,k-1,\rho)$. In $G(x,y)$, the edge labeled `insert` leads to node $(i(x,k-1,\rho)+1, i(y,k-1,\rho))$.
- `insert`: We have $\tau_k(x,\rho) =$ hash-insert, and $\tau_k(y,\rho) =$ hash-replace. Thus, $i(x,k,\rho) = i(x,k-1,\rho)$ and $i(y,k,\rho) = i(y,k-1,\rho)+1$. In $G(x,y)$, the edge labeled `insert` leads to node $(i(x,k-1,\rho), i(y,k-1,\rho)+1)$.
- `loop`: We have $\tau_k(x,\rho) =$ hash-insert, and $\tau_k(y,\rho) =$ hash-insert. Thus, $i(x,k,\rho) = i(x,k-1,\rho)$ and $i(y,k,\rho) = i(y,k-1,\rho)$. In $G(x,y)$, the edge labeled `loop` leads to node $(i(x,k-1,\rho), i(y,k-1,\rho))$. ◀

With this in mind, we can relate grid walks to hash collisions.

▶ **Lemma 8.** *Let x and y be any two strings, and ρ be any underlying function where both $\tau(x,\rho)$ and $\tau(y,\rho)$ are complete.*

Then $h_\rho(x) = h_\rho(y)$ if and only if $g(x,y,\rho)$ is alive. Furthermore, if $h_\rho(x) = h_\rho(y)$ then the path defined by $g(x,y,\rho)$ reaches node $(|x|,|y|)$.

Proof. *If direction*: Assume that $h_\rho(x) = h_\rho(y)$; we show that the path defined by $g(x,y,\rho)$ is alive and reaches $(|x|,|y|)$.

First, $g(x,y,\rho)$ must be alive: $g_k(x,y,\rho) = \texttt{stop}$ only when $x_{i(x,k,\rho)} \neq y_{i(y,k,\rho)}$ and either $\tau_k(x,\rho) =$ hash-match or $\tau_k(y,\rho) =$ hash-match, or when $k > \min\{|\tau(x,\rho)|, |\tau(y,\rho)|\}$. Since $x_{i(x,k,\rho)}$ (resp. $y_{i(y,k,\rho)}$) is appended to the hash on a hash-match, this contradicts $h_\rho(x) = h_\rho(y)$. Furthermore, we must have $|\tau(x,\rho)| = |\tau(y,\rho)|$ because $|\tau(x,\rho)| = |h_\rho(x)| = |h_\rho(y)| = |\tau(y,\rho)|$.

Since $\tau(x,\rho)$ and $\tau(y,\rho)$ are complete, $i(x,|\tau(x,\rho)|-1,\rho) = |x|$ and $i(y,\tau(y,\rho)-1,\rho) = |y|$. Thus, by Lemma 7, the walk reaches $(|x|,|y|)$.

Only If direction: We show that if $h_\rho(x) \neq h_\rho(y)$ then $g(x,y,\rho)$ is not alive. Let k be the smallest index such that the kth character of $h_\rho(x)$ is not equal to the kth character of $h_\rho(y)$. At least one of these characters cannot be $\perp$; thus either $\tau_k(x,\rho) =$ hash-match, or $\tau_k(y,\rho) =$ hash-match. If $x_{i(x,k,\rho)} \neq y_{i(y,k,\rho)}$, then $g_k(x,y,\rho) = \texttt{stop}$ and we are done. Otherwise, $x_{i(x,k,\rho)} = y_{i(y,k,\rho)}$; thus $\tau_k(x,\rho) = \tau_k(y,\rho)$, and the kth character of both $h_\rho(x)$ and $h_\rho(y)$ is $x_{i(x,k,\rho)} = y_{i(y,k,\rho)}$. But this contradicts the definition of k. ◀

We now bound the probability that the grid walk traverses each edge in $G(x,y)$.

▶ **Lemma 9.** *Let x and y be any two strings, and for any $k < 8d/(1-p_a) + 6\log n$ let E_k be the event that $i(x,k,\rho) < |x|$, $i(y,k,\rho) < |y|$, and $x_{i(x,k,\rho)} \neq y_{i(y,k,\rho)}$. Then if $\Pr_\rho[E_k] > 0$, the following four conditional bounds hold:*

$$\Pr_\rho[g_k(x,y,\rho) = \texttt{loop} \mid E_k] = p_a^2$$
$$\Pr_\rho[g_k(x,y,\rho) = \texttt{delete} \mid E_k] = p_a(1-p_a)p_r$$
$$\Pr_\rho[g_k(x,y,\rho) = \texttt{insert} \mid E_k] = p_a(1-p_a)p_r$$
$$\Pr_\rho[g_k(x,y,\rho) = \texttt{replace} \mid E_k] = (1-p_a)^2 p_r^2.$$

Proof. We have $|\tau(x,\rho)| > k$ and $|\tau(y,\rho)| > k$ from E_k. Thus:

- $\Pr_\rho(\tau_k(x,\rho) = \text{hash-insert} \mid E_k) = p_a$
- $\Pr_\rho(\tau_k(x,\rho) = \text{hash-replace} \mid E_k) = (1-p_a)p_r$
- $\Pr_\rho(\tau_k(x,\rho) = \text{hash-match} \mid E_k) = (1-p_a)(1-p_r)$.

The respective probabilities for $\tau_k(y,\rho)$ hold as well. Combining these probabilities with Table 1 gives the lemma. ◀

Transformations. We call a sequence of edits for a pair of strings x and y *greedy* if they can be applied to x in order from left to right, and all operations are performed on non-matching positions. We formally define this in Definition 10. With this in mind, we can simplify a sequence of edits for a given x and y, with the understanding that they will be applied greedily.

A *transformation* is a sequence of edits with position and character information removed: it is a sequence consisting only of `insert`, `delete`, and `replace`. We let $T(x,y)$ be the string that results from greedily applying the edits in T to x when x does not match y. We say that a transformation is *valid* for strings x and y if the total number of `delete` or `replace` operations in T is at most $|x|$, and the total number of `insert` or `replace` operations in T is at most $|y|$. The following definition formally defines how to apply these edits.

▶ **Definition 10.** *Let x and y be two \$-terminal strings, and let T be a transformation that is valid for x and y.*

If T is empty, $T(x,y) = x$. Otherwise we define $T(x,y)$ inductively. Let $T' = T[|T|-1]$ be T with the last operation removed, let $\sigma = T_{|T|-1}$ be the last operation in T, and let i be the smallest index such that the ith character of $T'(x,y)$ is not equal to y_i. Position i always exists if $T'(x,y) \neq y$ because x and y are \$-terminal; otherwise $i = 0$.[13]

We split into three cases depending on σ. If $\sigma =$ `insert`, we obtain $T(x,y)$ by inserting y_i at position i in $T'(x,y)$. If $\sigma =$ `delete`, we obtain $T(x,y)$ by deleting the ith character of $T'(x,y)$. Finally, if $\sigma =$ `replace`, we obtain $T(x,y)$ by replacing the ith character of $T'(x,y)$ with y_i.

We say that a transformation T *solves* x and y if T is valid for x and y, $T(x,y) = y$, and for any $i < |T|$, the prefix $T' = T[i]$ satisfies $T'(x,y) \neq y$.

A classic observation is that edit distance operations can be applied from left to right, greedily skipping all matches. The following lemma shows that this intuition applies to transformations. Since Definition 10 does not allow characters to be appended onto the end of x, we use the appended character \$ to ensure that there is an optimal transformation between any pair of strings.

[13] The case where i is reset to 0 is included for completeness and will not be used in the rest of the paper. It only occurs when x is first transformed into y, and then a sequence of redundant edits (such as an equal number of inserts and deletes) are performed.

▶ **Lemma 11.** *Let x and y be two strings that do not contain \$. Then if* $\mathrm{ED}(x, y) = r$,
- *there exists a transformation T of length r that solves $x \circ \$$ and $y \circ \$$, and*
- *there does not exist any transformation T' of length $< r$ that solves $x \circ \$$ and $y \circ \$$.*

Proof. We prove a single statement implying the lemma: if $\widehat{T}$ is the shortest transformation that solves $x \circ \$$ and $y \circ \$$, then $|\widehat{T}| = \mathrm{ED}(x, y)$.

Let $\sigma_1, \ldots \sigma_r$ be the sequence of edits applied to $x \circ \$$ to obtain $\widehat{T}(x \circ \$, y \circ \$)$ in Definition 10. These operations apply to increasing indices i because $\widehat{T}$ is the shortest transformation satisfying $\widehat{T}(x \circ \$, y \circ \$)$. Let σ_i be the last operation that applies to an index $i < |x|$. Let $\widehat{y} = \widehat{T}[i+1](x \circ \$, y \circ \$)$ be the string obtained after applying the operations of $\widehat{T}$ through σ_i. Clearly, x is a prefix of $\widehat{y}$. We claim that because $\widehat{T}$ is the shortest transformation, the operations in $\widehat{T}$ after σ_i must be $|\widehat{y}| - |x| - 1|$ `insert` operations. Clearly there must be at least $|\widehat{y} - |x| - 1|$ operations after σ_i because i is increasing and only one character in $\widehat{y}$ matches the final character \$ of x. By the same argument, if $\widehat{T}$ has any `insert` or `replace` operations it cannot meet this bound.

With this we have $\mathrm{ED}(x, y) \leq |\widehat{T}|$ because we can apply $\sigma_1, \ldots, \sigma_i$, followed by $|\widehat{y} - |x| - 1|$ insert operations to x to obtain y. This totals to $|\widehat{T}|$ operations overall.

We also have $\mathrm{ED}(x, y) \geq |\widehat{T}|$ by minimality of $\widehat{T}$ because any sequence of edits applied to x that obtains y will obtain $y \circ \$$ when applied to $x \circ \$$. ◀

For a given x, y, and ρ, we obtain the *transformation induced by x, y, and ρ*, denoted $\mathcal{T}(x, y, \rho)$, by removing all occurrences of `loop` and `match` from $g(x, y, \rho)$ if $g(x, y, \rho)$ is alive. Otherwise, $\mathcal{T}(x, y, \rho)$ is the empty string.

In Lemma 12 we show that strings x and y collide exactly when their induced transformation T solves x and y. This can be seen intuitively in Figure 2 – the grid walk is essentially a random walk through the dynamic programming table.

▶ **Lemma 12.** *Let x and y be two distinct strings and let $T = \mathcal{T}(x, y, \rho)$. Then $h_\rho(x) = h_\rho(y)$ if and only if T solves x and y.*

Proof. *If direction*: Assume T solves x and y. Since $x \neq y$, T must be nonempty; thus $g(x, y, \rho)$ is alive. By Lemma 8, $h_\rho(x) = h_\rho(y)$.

Only If direction: Assume $h_\rho(x) = h_\rho(y)$; by Lemma 8 $g(x, y, \rho)$ is alive.

Let $g(x, y, \rho)[k]$ be the prefix of $g(x, y, \rho)$ of length k, and let T^k be $g(x, y, \rho)[k]$ with `loop` and `match` removed. We prove by induction that $T^k(x[i(x, k, \rho)], y[i(y, k, \rho)]) = y[i(y, k, \rho)]$. This is trivially satisfied for $k = 0$.

Assume that $T^{k-1}(x[i(x, k-1, \rho)], y[i(y, k-1, \rho)]) = y[i(y, k-1, \rho)]$. We split into five cases based on the kth operation in $g(x, y, \rho)$.
- `match`: We must have $x_{i(x,k-1,\rho)} = y_{i(y,k-1,\rho)}$ and $T^k = T^{k-1}$. Furthermore, $i(x, k, \rho) = i(x, k-1, \rho) + 1$ and $i(y, k, \rho) = i(y, k-1, \rho) + 1$. Thus $T^k(x[i(x, k, \rho)], y[i(y, k, \rho)]) = T^{k-1}(x[i(x, k-1, \rho)], y[i(y, k-1, \rho)]) \circ x_{i(x,k-1,\rho)} = y[i(y, k, \rho)]$.
- `insert`: We have $i(x, k, \rho) = i(x, k-1, \rho)$ and $i(y, k, \rho) = i(y, k-1, \rho) + 1$. Thus, $T^{k-1}(x[i(x, k, \rho)], y[i(y, k, \rho)]) = y[i(y, k-1, \rho)]$ and $y[i(y, k, \rho)]$ differ only in the last character. Then $T^k(x[i(x, k, \rho)], y[i(y, k, \rho)]) = T^{k-1}(x[i(x, k-1, \rho)], y[i(y, k-1, \rho)]) \circ y_{i(y,k,\rho)-1} = y[i(y, k, \rho)]$.
- `replace`: We have $i(x, k, \rho) = i(x, k-1, \rho) + 1$ and $i(y, k, \rho) = i(y, k-1, \rho) + 1$. Thus, $T^{k-1}(x[i(x, k, \rho)], y[i(y, k, \rho)]) = y[i(y, k-1, \rho)] \circ x_{i(x,k,\rho)-1}$ and $y[i(y, k, \rho)]$ differ only in the last character. By definition, the final character of $T^{k-1}(x[i(x, k, \rho)], y[i(y, k, \rho)]$ is replaced with $y_{i(y,k,\rho)-1}$, obtaining $T^k(x[i(x, k, \rho)], y[i(y, k, \rho)]) = y[i(y, k-1, \rho)] \circ y_{i(y,k,\rho)-1} = y[i(y, k, \rho)]$.

- `delete`: We have $i(x,k,\rho) = i(x,k-1,\rho)+1$ and $i(y,k,\rho) = i(y,k-1,\rho)$. Thus, $T^{k-1}(x[i(x,k,\rho)], y[i(y,k,\rho)]) = y[i(y,k-1,\rho)] \circ x_{i(x,k,\rho)-1}$ and $y[i(y,k,\rho)]$ differ only in the last character (which is deleted). Then $T^k(x[i(x,k,\rho)], y[i(y,k,\rho)]) = T^{k-1}(x[i(x,k-1,\rho)], y[i(y,k-1,\rho)]) = y[i(y,k,\rho)]$.
- `loop`: We have $i(x,k,\rho) = i(x,k-1,\rho)$, $i(y,k,\rho) = i(y,k-1,\rho)$, and $T^k = T^{k-1}$. We immediately obtain $T^k(x[i(x,k,\rho)], y[i(y,k,\rho)]) = y[i(y,k,\rho)]$.

By Lemma 8, $g(x,y,\rho)$ reaches node $(|x|,|y|)$, so the above shows that with $k = |g(x,y,\rho)|$, $T(x,y) = y$. ◀

We are finally ready to prove Lemma 13, which forms the basis of our performance analysis.

▶ **Lemma 13.** *For any \$-terminal strings x and y, let T be a transformation of length t that is valid for x and y. Then*

$$p^t - 1/n^2 \le \Pr_\rho[T \text{ is a prefix of } \mathcal{T}(x,y,\rho)] \le p^t.$$

Proof. Define a *grid sequence* to be a sequence of grid walk operations.

Let G_T be the set of all grid sequences g such that g does not contain `stop`, and deleting `loop` and `match` from g results in a transformation T_g such that T is a prefix of T_g. Then by definition, if T is a prefix of $\mathcal{T}(x,y,\rho)$ then $g(x,y,\rho) \in G_T$; furthermore, if $g(x,y,\rho) \in G_T$ and $g(x,y,\rho)$ is complete, then T is a prefix of $\mathcal{T}(x,y,\rho)$.

We begin by proving that $\Pr_\rho[g(x,y,\rho) \in G_T] = p^t$. We prove this by induction on t; $t = 0$ is trivially satisfied. We assume that for any transformation T' of length $|T'| = t-1$, we have $\Pr_\rho[g(x,y,\rho) \in G_{T'}] = p^{t-1}$, and prove the above for any T with $|T| = t$.

Let σ be the last operation in T, and let $T' = T[|T|-1]$ be T with σ removed. Thus, $g(x,y,\rho) \in G_T$ only if there exist (possibly empty) grid sequences g' and g'' satisfying

- $g' \in G_{T'}$,
- g'' consists of `loop` and `match` operations concatenated onto the end of $g' \circ \sigma$, ending with a `match` operation if g'' is nonempty, and
- $g(x,y,\rho)$ consists of zero or more `loop` operations concatenated onto g''.

By definition of conditional probability,

$$\Pr[g(x,y,\rho) \in G_T] = \sum_{g'} \left(\Pr[g' \in G_{T'}] \cdot \sum_{g''} \Pr[g'' \in G_T \mid g' \in G_{T'}] \Pr[g(x,y,\rho) \in G_T \mid g'' \in G_T] \right).$$

We bound these terms one at a time.

Clearly there is only one g' satisfying the conditions, which can be obtained by taking the prefix of $g(x,y,\rho)$ before the final `insert`, `delete`, or `replace` operation. By the inductive hypothesis, $\sum_{g'} \Pr[g' \in G_{T'}] = p^{t-1}$.

We now bound $\Pr[g'' \in G_T \mid g' \in G_{T'}]$. The conditional means that we can invoke Lemma 9 (as $\Pr[E_k] = p^{t-1} > 0$).

We have $\Pr[g'' \in G_T \mid g' \in G_{T'}] = \Pr[g'' \in G_T \mid g' \circ \sigma \in G_T] \Pr[g' \circ \sigma \in G_T \mid g' \in G_{T'}]$.

We split into two cases depending on σ. Recall that $p_r = p_a/(1-p_a)$. Since T is valid, if $\sigma =$ `delete` or $\sigma =$ `replace` we cannot have $i(x,k,\rho) = |x|-1$; similarly if $\sigma =$ `insert` or $\sigma =$ `replace` we cannot have $i(y,k,\rho) = |y|-1$. Then by Lemma 9, if $\sigma =$ `delete` or $\sigma =$ `insert`, $\Pr(g' \circ \sigma \in G_T \mid g' \in G_{T'}) = p_a(1-p_a)p_r = p_a^2$. Similarly, if $\sigma =$ `replace`, $\Pr(g' \circ \sigma \in G_T \mid g' \in G_{T'}) = (1-p_a)^2 p_r^2 = p_a^2$. For any k such that $i(x,k,\rho) = i(y,k,\rho)$, $g_k(x,y,\rho) \neq$ `stop` by definition; meanwhile, if $i(x,k,\rho) \neq i(y,k,\rho)$ then $g_k(x,y,\rho) \neq$ `match`. Thus, $\sum_{g''} \Pr(g'' \in G_T \mid g' \circ \sigma \in G_T) = 1$.

Finally we bound $\Pr[g(x,y,\rho) \in G_T \mid g'' \in G_T]$. Let ℓ be the number of operations concatenated onto g'' to obtain $g(x,y,\rho)$. Then by Lemma 9,

$$\Pr[g(x,y,\rho) \in G_T \mid g'' \in G_T] = \sum_{\ell} p_a^{2\ell} = 1/(1-p_a^2).$$

Multiplying the above bounds, we have $\Pr[g(x,y,\rho) \in G_T] = p^{t-1}p_a^2/(1-p_a^2)$. Noting that $p = p_a^2/(1-p_a^2)$, we obtain $\Pr[g(x,y,\rho) \in G_T] = p^t$.

We have that if T is a prefix of $\mathcal{T}(x,y,\rho)$ then $g(x,y,\rho) \in G_T$; thus

$$\Pr_{\rho}[T \text{ is a prefix of } \mathcal{T}(x,y,\rho)] \leq p_t.$$

Meanwhile, T is a prefix of $\mathcal{T}(x,y,\rho)$ if $g(x,y,\rho) \in G_T$ and $g(x,y,\rho)$ is complete. By the inclusion-exclusion principle,

$$\begin{aligned}\Pr[T \text{ is a prefix of } \mathcal{T}(x,y,\rho)] &= \Pr[g(x,y,\rho) \in G_T] + \Pr[g(x,y,\rho) \text{ is complete}] - \\ &\Pr[g(x,y,\rho) \in G_T \text{ or } g(x,y,\rho) \text{ is complete}] \geq p^t + \Pr[g(x,y,\rho) \text{ is complete}] - 1.\end{aligned}$$

We have that $\Pr[g(x,y,\rho) \text{ is complete}] = 1 - \Pr[\tau(x,\rho) \text{ or } \tau(y,\rho) \text{ is not complete}]$. By union bound and Lemma 6, $\Pr[g(x,y,\rho) \text{ is complete}] \geq 1 - 2/n^2$. Substituting,
$\Pr[T \text{ is a prefix of } \mathcal{T}(x,y,\rho)] \geq p^t - 2/n^2$. ◀

4.2 Bounds on Collision Probabilities

We can now bound the probability that two strings collide.

▶ **Lemma 14.** *If x and y satisfy $\mathrm{ED}(x,y) \leq r$, then $\Pr_{\rho}(h_\rho(x) = h_\rho(y)) \geq p^r - 2/n^2$.*

Proof. Because $\mathrm{ED}(x,y) \leq r$, by Lemma 11 there exists a transformation T of length r that solves x and y. By Lemma 13, h induces T on x and y (which is sufficient for $h(x) = h(y)$ by Lemma 12) with probability $p^r - 2/n^2$. ◀

The corresponding upper bound requires that we sum over many possible transformations.

▶ **Lemma 15.** *If x and y satisfy $\mathrm{ED}(x,y) \geq cr$, then $\Pr_{\rho}(h_\rho(x) = h_\rho(y)) \leq (3p)^{cr}$.*

Proof. Let $\mathcal{T}$ be the set of all transformations that solve x and y. By Lemma 12 and Lemma 13,

$$\Pr_{h \in \mathcal{H}}(h(x) = h(y)) = \sum_{T \in \mathcal{T}} p^{|T|}.$$

Thus, we want to find the $\mathcal{T}$ (for the given x and y) that maximizes this probability.

Since all pairs $T_1, T_2 \in \mathcal{T}$ solve x and y, there is no pair $T_1, T_2 \in \mathcal{T}$ such that T_1 is a prefix of T_2. Thus, $\mathcal{T}$ can be viewed as the leaves of a trie of branching factor at most 3, where each leaf has depth at least cr.

We show that without loss of generality all leaves are at depth cr. Consider a leaf T_1 at the maximum depth of the trie $i > cr$, and its siblings T_2 and T_3 if they exist. Collapse this leaf and its siblings, replacing them instead with a leaf T_p corresponding to their parent in the trie; call the resulting set $\mathcal{T}'$. Since we have added a transformation of length $i-1$ and removed at most three of length i, this changes the total cost of $\mathcal{T}$ by at least $p^{i-1} - 3p^i$; this is positive since $p \leq 1/3$. Repeating this process results in a set $\mathcal{T}_M$ with all nodes at depth cr, where $\mathcal{T}_M$ gives larger collision probability than the original set $\mathcal{T}$.

There are at most 3^{cr} transformations in $\mathcal{T}_M$, each of length cr. Thus $\Pr_{h \in \mathcal{H}}(h(x) = h(y)) \leq 3^{cr}p^{cr}$. ◀

The following special case is not used in our similarity search bounds, but may be useful in understanding performance on some datasets. In short, strings that do not have any matching characters achieve better performance bounds. It would be interesting to see if this analysis can be extended to other special cases.

▶ **Lemma 16.** *Let x and y be two \$-terminal strings with $\mathrm{ED}(x,y) \geq cr$ such that for all $i < |x| - 1$ and $j < |y| - 1$, $x_i \neq y_j$. Then $\Pr_\rho(h_\rho(x) = h_\rho(y)) \leq (2p/(1-p))^{cr}$.*

Proof. Let $\widehat{x}$ and $\widehat{y}$ be arbitrary \$-terminal strings of length cr with no other characters in common. We use grid walks on $G(\widehat{x},\widehat{y})$ to reason about grid walks on $G(x,y)$.

Let $GR(i,j)$ be the set of all grid walks reaching node (i,j) in $G(\widehat{x},\widehat{y})$. Let $W(i,j) = \Pr[g(\widehat{x},\widehat{y},\rho) \in GR(i,j)]$. We have $W(0,0) = 1$.

Clearly, $GR(i,j)$ is a subset of $GR(i-1,j) \cup GR(i-1,j-1) \cup GR(i,j-1)$. In fact, using a case-by-case analysis essentially identical to that of Lemma 13,

$$W(i,j) \leq p \cdot W(i-1,j) + p \cdot W(i-1,j-1) + p \cdot W(i,j-1).$$

We take $W(i^*,-1) = 0 = W(-1,j^*)$ for all i^* and j^* so that we can state this recursion without border cases.

We show by induction that if $\max i,j = \ell$, then $W(i,j) \leq (2p/(1-p))^\ell$. This is already satisfied for $\ell = 0$.

Assume that the induction is satisfied for all $W(i^*,j^*)$ with $\max\{i^*,j^*\} = \ell - 1$. For all (i,j) such that $\max\{i,j\} = \ell$, at most two of $(i-1,j-1)$, $(i-1,j)$, and $(i,j-1)$ have max $\ell - 1$; the remaining pair has max ℓ. Thus

$$W(i,j) \leq p\left(\frac{2p}{1-p}\right)^{\ell-1} + p\left(\frac{2p}{1-p}\right)^{\ell-1} + p\left(\frac{2p}{1-p}\right)^{\ell} \leq \left(\frac{2p}{1-p}\right)^{\ell}$$

All grid walks in $G(x,y)$ that go through $(|x|-1,|y|-1)$ must be in $GR(|x|-1,|y|-1)$. Since we must have $\max\{|x|,|y|\} = cr+1$, the proof is complete. ◀

4.3 Final Running Time for Approximate Similarity Search

In this section, we describe how to get from our LSH to an algorithm satisfying Definition 3, proving Theorem 1.

Space and Preprocessing. To preprocess, we first pick $R = \Theta(1/p_1)$ underlying hash functions $\rho_1, \rho_2, \ldots, \rho_R$. For each string x stored in the database, we calculate $h_{\rho_1}(x), \ldots, h_{\rho_R}(x)$, and store them in a dictionary data structure for fast lookups (for example, these can be stored in a hash table, where each $h_\rho(x)$ has a back pointer to x). We set $1/p_1 = 3^r n^{1/c}$ (see the discussion below), leading to space $\widetilde{O}(d3^r n^{1+1/c})$.

We store the underlying functions $\rho_1, \rho_2, \ldots \rho_R$ so they can be used during queries. In Section 4.5, we show that these functions can be stored in $\widetilde{O}(|\Sigma| dR)$ space, which is a lower order term if $|\Sigma| = O(n)$.

In the common case that $|\Sigma| = O(n/d)$, the underlying functions are cheap to store, and we can further decrease the space. For each x, we can store a random $\log n$-bit hash of $h_\rho(x)$ for all ρ, rather than the full hash string of length $\Theta(d)$. This gives a space bound of $\widetilde{O}(3^r n^{1+1/c} + dn)$.

Queries. For a given query q, we calculate $h_1(q), h_2(q), \ldots, h_R(q)$. For each database string x that collides with q (i.e. for each x such that there exists an i with $h_{\rho_i}(q) = h_{\rho_i}(x)$), we calculate $\mathrm{ED}(x, q)$. We return x if the distance is at most cr. After repeating this for all R underlying functions, we return that there is no close point.

Correctness of the data structure follows from the definition of p_1: if $\mathrm{ED}(q, x) \leq r$, then after $\Theta(1/p_1)$ independent hash functions, q and x collide on at least one hash function with constant probability.

The cost of each repetition is the cost to hash, plus the number of database elements at distance $> cr$ that collide with q. The cost to hash is $O(d/(1-p_a) + \log n)$ by definition, and the cost to test if two strings have distance at most cr is $O(dcr)$ by [36]. The number of elements with distance $> cr$ that collide with q is at most np_2 in expectation. Thus our total expected cost can be written

$$O\left(\frac{1}{p_1}\left(\frac{d}{1-p_a} + \log n + (dcr)np_2\right)\right).$$

This can be minimized (up to a factor $O(\log n)$) by setting $p_2 = 1/ncr$ (recall that $p_a \leq 1/2$).

Thus, we set $p_2 = 1/ncr$, which occurs at $p = 1/(3(ncr)^{1/cr})$. Using this value of p, we get $p_1 \geq p^r = \Omega(1/(r3^r n^{1/c}))$.

Putting this all together, the expected query time is $\widetilde{O}(d3^r n^{1/c})$.

4.4 Approximate Nearest Neighbor

In this section we generalize Section 4.3 to prove Theorem 2. Let $R = \{i \in \{1, \ldots, d\} \mid 3^i n^{1/c} \leq n\}$. We build $O(\log n)$ copies of the data structure described in Section 4.3 for each $r^* \in R$.

Queries. We iterate through each $r^* \in R$ in increasing order, querying the data structure as described above. If we find a string at distance at most cr^* we stop and return it. If we reach an r^* such that $3^{r^*} n^{1/c} > n$, we simply scan through all strings to check which is the closest.

Assume the actual nearest neighbor is at distance r. By Chernoff bounds, we succeed with high probability when $r^* = r$; that is, we return a string at distance at most cr. Thus, the cost is at most $\sum_{r^*=1}^{r} \widetilde{O}(d3^{r^*} n^{1/c}) = \widetilde{O}(d3^r n^{1/c})$ with high probability.

Space. We build $O(\log n)$ copies of each data structure; thus the total space is $\sum_{r=1}^{r^*} \widetilde{O}(d3^r n^{1+1/c}) = \widetilde{O}(dn^2)$ by definition of r^*. We obtain preprocessing time $\widetilde{O}(dn^2)$ immediately.

4.5 Storing Underlying Functions

Our algorithm uses a large number of fully-random, real-number hashes; this causes issues with the space bounds since we need to store each hash. In this section we relax this assumption.

We modify ρ to hash to a uniformly random element of the set $\{0, \epsilon, 2\epsilon, \ldots, 1\}$. Since the domain of each ρ has size $O(|\Sigma|(d + \log n)$, this means that each ρ can be stored in $O(|\Sigma| \log(1/\epsilon)(d + \log n))$ bits of space.

Intuitively, setting $\epsilon = 1/n$ should not affect our query bounds, while still retaining the space bounds of Theorems 1 and 2. We show this in Lemma 17.

▶ **Lemma 17.** *With p_a and p_r increased by $\epsilon = 1/n$, and assuming $d = O(n)$, if x and y satisfy $\mathrm{ED}(x,y) \le r$, then $\Pr(h(x) = h(y) \ge \Omega(p^r - 2/n^2)$. If x' and y' satisfy $\mathrm{ED}(x',y') \ge cr$ then $\Pr((h(x) = h(y)) \le O((3p)^{cr})$.*

Proof. For simplicity, we let $\widehat{p_a} = p_a + \epsilon$ and $\widehat{p_r} = p_r + \epsilon$.

Since $p_1 = \Omega(1/(r3^r n^{1/c})$, we have (omitting constants for simplicity) $p = 1/(3n^{1/rc})$. Therefore, $p_a = \sqrt{1/(1+3n^{1/rc})} \gg 1/n$, and thus $p_r = p_a/(1-p_a) \gg 1/n$. Thus, $p_a < \widehat{p_a} < p_a(1+1/n)$ and $p_r < \widehat{p_r} < p_r(1+1/n)$.

Let ϵ' satisfy

$$p(1-\epsilon') \le \frac{\widehat{p_a}(1-\widehat{p_a})\widehat{p_r}}{(1-\widehat{p_a}^2)} \le p(1+\epsilon') \quad \text{and} \quad p(1-\epsilon') \le \frac{(1-\widehat{p_a})^2\widehat{p_r}^2}{(1-\widehat{p_a}^2)} \le p(1+\epsilon'). \quad (1)$$

Then the proof of Lemma 13 gives that for any \$-terminal strings x and y, and any transformation T of length t,

$$(p(1-\epsilon'))^t - 1/n^2 \le \Pr_{\rho}[T \text{ is a prefix of } \mathcal{T}(x,y,\rho)] \le (p(1+\epsilon'))^t.$$

So long as $(1 \pm \epsilon')^t = \Theta(1)$ we are done. Clearly this is the case for $\epsilon' = O(1/n)$ since $t \le 2d = O(n)$. We prove each bound in Equation (1) one term at a time for $\epsilon' = O(1/n)$. First inequality (recall that $p_a \le 1/2$):

$$\frac{\widehat{p_a}(1-\widehat{p_a})\widehat{p_r}}{(1-\widehat{p_a}^2)} > \frac{p_a(1-p_a(1+1/n))p_r}{1-p_a^2} = p - \frac{p_a p_r}{n(1-p_a^2)} = p - \frac{p}{n(1-p_a)} = p(1-O(n))$$

Second inequality:

$$\begin{aligned}\frac{\widehat{p_a}(1-\widehat{p_a})\widehat{p_r}}{(1-\widehat{p_a}^2)} &< \frac{p_a(1+1/n)^2(1-p_a)p_r}{1-(p_a(1+1/n))^2} = \frac{p_a(1-p_a)p_r}{1/(1+1/n)^2 - p_a^2} \\ &= \frac{p_a(1-p_a)p_r}{1-O(1/n^2)-p_a^2} < \frac{p_a(1-p_a)p_r}{(1-p_a^2)(1-O(1/n^2))} \\ &= p(1+O(1/n^2))\end{aligned}$$

Third inequality (since $p_a \le 1/2$, $2p_a \le 4(1-p_a)^2$):

$$\begin{aligned}\frac{(1-\widehat{p_a})^2\widehat{p_r}^2}{(1-\widehat{p_a}^2)} &\ge \frac{(1-p_a(1+1/n))^2 p_r^2}{(1-{p_a}^2)} \\ &= \frac{(1-2p_a(1+1/n)+p_a^2(1+1/n)^2)p_r^2}{(1-{p_a}^2)} \\ &> \frac{((1-p_a)^2 - 2p_a/n)p_r^2}{(1-{p_a}^2)} = p - \frac{2p_a p_r^2}{n(1-{p_a}^2)} \\ &\ge p(1-O(1/n))\end{aligned}$$

Fourth inequality (this is largely the same as the second inequality):

$$\frac{(1-\widehat{p_a})^2\widehat{p_r}^2}{(1-\widehat{p_a}^2)} \le \frac{(1-p_a)^2 p_r(1+1/n)^2}{1-(p_a(1+1/n))^2} = p(1+O(1/n^2))$$ ◀

References

1. Thomas Dybdahl Ahle, Martin Aumüller, and Rasmus Pagh. Parameter-free locality sensitive hashing for spherical range reporting. In *Proc. 28th Symposium on Discrete Algorithms (SODA)*, pages 239–256. SIAM, 2017.
2. Josh Alman and Ryan Williams. Probabilistic polynomials and Hamming nearest neighbors. In *Proc. 56th Symposium on Foundations of Computer Science (FOCS)*, pages 136–150. IEEE, 2015.
3. Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. In *Proc. International Congress of Mathematicians (ICM)*, pages 3271–3302, 2018.
4. Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proc. 28th Symposium on Discrete Algorithms (SODA)*, pages 47–66. ACM-SIAM, 2017.
5. Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 2019.
6. Leonid Boytsov. Indexing methods for approximate dictionary searching: Comparative analysis. *Journal of Experimental Algorithmics (JEA)*, 16:1–1, 2011.
7. Eric Brill and Robert C Moore. An improved error model for noisy channel spelling correction. In *Proc. 38th Meeting on Association for Computational Linguistics*, pages 286–293. Association for Computational Linguistics, 2000.
8. Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proc. 48th Annual Symposium on Theory of Computing (STOC)*, pages 712–725. ACM, 2016.
9. Ho-Leung Chan, Tak-Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. A linear size index for approximate pattern matching. In *Proc. 17th Symposium on Combinatorial Pattern Matching (CPM)*, pages 49–59. Springer, 2006.
10. Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. 34th Symposium on Theory of Computing (STOC)*, pages 380–388. ACM, 2002.
11. Flavio Chierichetti, Ravi Kumar, and Mohammad Mahdian. The complexity of LSH feasibility. *Theoretical Computer Science*, 530:89–101, 2014.
12. Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *Proc. 28th Symposium on Discrete Algorithms (SODA)*, pages 31–46. Society for Industrial and Applied Mathematics, 2017.
13. Tobias Christiani and Rasmus Pagh. Set similarity search beyond minhash. In *Proc. 49th Symposium on Theory of Computing (STOC)*, pages 1094–1107. ACM, 2017.
14. Vincent Cohen-Addad, Laurent Feuilloley, and Tatiana Starikovskaya. Lower bounds for text indexing with mismatches and differences. In *Proc. 30th Symposium on Discrete Algorithms (SODA)*, pages 1146–1164. ACM-SIAM, 2019.
15. Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proc. 36th ACM Symposium on Theory of Computing (STOC)*, pages 91–100. ACM, 2004.
16. Benjamin Coleman, Richard Baraniuk, and Anshumali Shrivastava. Sub-linear memory sketches for near neighbor search on streaming data. In *Proc. 27th International Conference on Machine Learning (ICML)*, pages 2089–2099. PMLR, 2020.
17. Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proc. 20th Conference on the World Wide Web (WWW)*, pages 577–586. ACM, 2011.
18. Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.
19. Piotr Indyk. Approximate nearest neighbor under edit distance via product metrics. In *Proc. 15th Symposium on Discrete Algorithms (SODA)*, pages 646–650. ACM-SIAM, 2004.

20 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. 30th Symposium on Theory of Computing (STOC)*, pages 604–613. ACM, 1998.

21 Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

22 Tamer Kahveci, Vebjorn Ljosa, and Ambuj K Singh. Speeding up whole-genome alignment by indexing frequency vectors. *Bioinformatics*, 20(13):2122–2134, 2004.

23 Subhash Khot and Assaf Naor. Nonembeddability theorems via fourier analysis. *Mathematische Annalen*, 334(4):821–852, 2006.

24 Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, Chi-Kwong Wong, and Siu-Ming Yiu. Compressed indexing and local alignment of dna. *Bioinformatics*, 24(6):791–797, 2008.

25 Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *Transactions on Knowledge and Data Engineering (TKDE)*, 32(8):1475–1488, 2019.

26 Moritz G Maaß and Johannes Nowak. Text indexing with errors. In *Proc. 16th Symposium on Combinatorial Pattern Matching (CPM)*, pages 21–32. Springer, 2005.

27 Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2020.

28 Udi Manber and Sun Wu. An algorithm for approximate membership checking with application to password security. *Information Processing Letters*, 50(4):191–197, 1994.

29 Guillaume Marçais, Dan DeBlasio, Prashant Pandey, and Carl Kingsford. Locality sensitive hashing for the edit distance. *Bioinformatics*, 35(14):i127–i135, 2019.

30 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

31 Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

32 Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *Journal of the ACM (JACM)*, 54(5):23, 2007.

33 Ozgur Ozturk and Hakan Ferhatosmanoglu. Effective indexing and filtering for similarity search in large biosequence databases. In *Proc. 3rd Symposium on Bioinformatics and Bioengineering*, pages 359–366. IEEE, 2003.

34 Rasmus Pagh, Ninh Pham, Francesco Silvestri, and Morten Stöckel. I/O-efficient similarity join. *Algorithmica*, 78(4):1263–1283, 2017.

35 Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *Proc. 50th Symposium on Theory of Computing (STOC)*, pages 1260–1268. ACM, 2018.

36 Esko Ukkonen. Algorithms for approximate string matching. *Information and control*, 64(1-3):100–118, 1985.

37 Yiqiu Wang, Anshumali Shrivastava, Jonathan Wang, and Junghee Ryu. Randomized algorithms accelerated over cpu-gpu for ultra-high dimensional similarity search. In *Proc. International Conference on Management of Data (SIGMOD)*, pages 889–903. ACM, 2018.

38 W John Wilbur, Won Kim, and Natalie Xie. Spelling correction in the pubmed search engine. *Information retrieval*, 9(5):543–564, 2006.

39 Haoyu Zhang and Qin Zhang. Embedjoin: Efficient edit similarity joins via embeddings. In *Proc. 23rd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 585–594. ACM, 2017.

40 Haoyu Zhang and Qin Zhang. Minjoin: Efficient edit similarity joins via local hash minima. In *Proc. 25th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1093–1103. ACM, 2019.

Locality-Aware Distribution Schemes

Bruhathi Sundarmurthy
University of Wisconsin-Madison, Madison, WI, USA

Paraschos Koutris
University of Wisconsin-Madison, Madison, WI, USA

Jeffrey Naughton
University of Wisconsin-Madison, Madison, WI, USA

Abstract

One of the bottlenecks in parallel query processing is the cost of shuffling data across nodes in a cluster. Ideally, given a distribution of the data across the nodes and a query, we want to execute the query by performing only local computation and no communication: in this case, the query is called parallel-correct with respect to the data distribution. Previous work studied this problem for Conjunctive Queries in the case where the distribution scheme is oblivious, i.e., the location of each tuple depends only on the tuple and is independent of the instance. In this work, we show that oblivious schemes have a fundamental theoretical limitation, and initiate the formal study of distribution schemes that are locality-aware. In particular, we focus on a class of distribution schemes called co-hash distribution schemes, which are widely used in parallel systems. In co-hash partitioning, some tables are initially hashed, and the remaining tables are co-located so that a join condition is always satisfied. Given a co-hash distribution scheme, we formally study the complexity of deciding various desirable properties, including obliviousness and redundancy. Then, for a given Conjunctive Query and co-hash scheme, we determine the computational complexity of deciding whether the query is parallel-correct. We also explore a stronger notion of correctness, called parallel disjoint correctness, which guarantees that the query result will be disjointly partitioned across nodes, i.e., there is no duplication of results.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory)

Keywords and phrases partitioning, parallel correctness, join queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.22

Funding *Paraschos Koutris*: NSF CRII-1850348.

1 Introduction

Modern data management systems utilize the power of parallelism to efficiently process huge datasets. These systems can often scale to hundreds, and even thousands of machines. However, as the data and scale increases, massively parallel systems face a critical bottleneck: the cost of communicating (or shuffling) data across different machines. The amount of data shuffling required to process a given query depends on the initial data distribution, or data partitioning. Given a query, it is desirable to obtain the query result with the minimum possible shuffling of data, while making sure that no machine is overloaded. Ideally, we can execute the query without any shuffling, by simply running it on the local fragment of each machine/node, and then taking the union of all the results. This notion of being able to execute queries with no data shuffling is called *parallel correctness*, or p-correctness for short, and was first introduced in [3].

Previous works [3, 4, 9, 12, 13, 18] have studied the problem of p-correctness for different classes of queries, including Conjunctive Queries (CQs), Unions of Conjunctive Queries (UCQs), and Datalog programs. In all prior work, the data distribution is specified by a *distribution policy*, where the location of each tuple depends only on the tuple, and not on

the rest of the database. A distribution policy can hence be captured by a function $\mathbf{P}(t)$ that maps a tuple t to a set of machines/nodes. Typical examples of distribution policies are hash partitioning, where each relation in the database is hashed on a chosen subset of its attributes, as well as hypercube partitioning [2, 3, 8, 12]. The latter distribution policy is used for load-optimal single round algorithms that compute CQs [5, 15], when the initial data distribution is arbitrary and thus any correct algorithm must take into account that there is no knowledge of where the data lies.

In this paper, we consider a more general class of distribution schemes, captured by a function $\mathbf{P}(t, I)$ that allocates a tuple using (possibly) additional information from the instance. Distribution schemes that ignore I are called *oblivious*, and correspond to distribution policies. Oblivious distribution schemes, although simple to understand, can often lead to suboptimal partitioning, since the locality of the data across relations (or within the same relation) is not exploited at all. As we will show in Section 3, any oblivious scheme that guarantees p-correctness for some query q incurs a certain storage overhead that is unavoidable regardless of the underlying instance.

To overcome this barrier, it is necessary to look into distribution schemes that are *locality-aware*, in the sense that tuples are distributed taking into account tuples in other relations. In practice, parallel data management systems partition data in smart ways [16, 17, 21] and several parallel large-scale systems are deploying locality-aware data partitionings [7, 19, 20, 22, 23] to minimize, or even reduce to zero, the amount of data shuffling for a given query or query workload.

Since it is generally infeasible to describe efficiently a non-oblivious distribution scheme, in practice we are interested in schemes that can be concisely represented. In this work, we will focus on a particular class of locality-aware distribution schemes that has been widely adopted in practice called *co-hashing*. In such a scheme, a set of relations is initially hash-partitioned. The tuples from the remaining relations are then collocated with tuples from other relations according to specified join conditions. We illustrate co-hashing with an example below:

▶ **Example 1.** Consider two binary relations $R(A_1, A_2)$ and $S(B_1, B_2)$. A co-hash scheme partitions R by hashing on attribute A_1, and then distributes each tuple s from S to all nodes that consist of a tuple r from R that joins with s on the join condition $R.A_2 = S.B_1$. The tuples of S that do not join with any tuple in R are hashed on attribute B_1. The scheme is locality-aware since the nodes where a tuple from S is assigned depend on the tuples that occur in R. It is easy to observe that, given the above co-hash scheme, the query $q(x, y, z) = R(z, x), S(x, y)$ is parallel-correct.

In a general co-hash scheme, we can chain together arbitrarily many relations, as long as we are not introducing a cyclic dependency. Note also that a co-hash scheme strictly generalizes (oblivious) schemes where each relation is hashed independently. Though co-hashing can make join processing very efficient, determining whether a query is parallel-correct for a given co-hash distribution scheme can be challenging, as illustrated in the example below:

▶ **Example 2.** Consider the setup from Example 1, together with a third table $U(D_1)$ that is co-hashed with R on $R.A_2 = U.D_1$. Let the tuples of U that do not join with R be hashed on attribute D_1. Now, consider the Conjunctive Query $q(x, y) = S(x, y), U(x)$. As we will see later, this query is parallel correct for this co-hash scheme for any input instance.

Consider the naive approach of checking p-correctness for a CQ: look at one binary join at-a-time, and check if this join appears somewhere in the co-hash scheme. This approach is employed by the current state of the art technique [23], but it fails in this example, since

S, U are not directly connected in the co-hash scheme (but only through R). Hence, the system will conclude that data shuffling must be performed, even if it is not necessary. We should note that p-correctness depends critically on how the non-joining tuples are handled. For instance, if we choose to hash the non-joining tuples from S on attribute B_2 instead, q is not p-correct any more.

The notion of p-correctness guarantees that the union of the local results reconstructs the query result. However, a query result may end up in multiple nodes, so obtaining the final result may require an additional deduplication step. If we want to avoid this, we need to check for a stronger condition, *parallel disjoint correctness*, or pd-correctness, which determines whether tuples of a query result are disjointly partitioned across the nodes.

In addition to the correctness notions, we introduce and study in the context of co-hash schemes two additional properties. The first property, *obliviousness*, tests whether a particular relation can be oblivious in an otherwise non-oblivious scheme. This is important, because such a relation can be easily maintained in a parallel setting. The second property, *clustering*, tests whether tuples of a given relation that agree on a set of attributes are always located in the same unique node. For example, relation R from Example 1 is clustered w.r.t. attribute A_1. Note that this implies that any query that groups by A_1 and aggregates can be computed locally without any data shuffling. Clustering with respect to all attributes in a relation R is equivalent to *non-redundancy* (i.e., each tuple in R is assigned to a unique node). For instance, relation S from Example 1 exhibits redundancy, since an S-tuple can end up in two (or more) nodes.

Our Contributions. In this paper, we study the formal foundations for locality-aware distribution schemes. We next summarize our contributions.

1. **Theoretical Framework.** We introduce a general framework that captures locality-aware distribution schemes. We describe several desirable properties, including p-correctness, pd-correctness, obliviousness and clustering. Within this framework, we show that oblivious schemes have a fundamental barrier on how well they can localize data.
2. **Co-hash Schemes.** We formalize the class of co-hash distribution schemes, which is widely used in practice, by introducing the notion of a *co-hash graph* to concisely capture how the input instance is distributed.
3. **Deciding Properties**: We study three properties of co-hash schemes: balancedness, obliviousness and clustering. We first show how co-hash schemes can overcome the limitations of oblivious schemes. Then, we show that we can decide both obliviousness and clustering in polynomial time (in the size of the co-hash graph). We observe that taking functional dependencies into account can lead to better reasoning about whether a relation is clustered or not.
4. **Parallel correctness**: We study the complexity of determining parallel correctness for co-hash schemes for the class of CQs. We distinguish two subproblems, depending on whether we consider a specific instance, or we want to determine p-correctness across all possible instances. We show that the former subproblem is Π_P^2-complete for CQs, but coNP-complete when restricted to full CQs (CQs without projections). For the latter subproblem, we show that it is NP-hard for general CQs, while for full CQs the complexity drops to polynomial time.
5. **Parallel disjoint correctness**: Finally, we provide results for the complexity of pd-correctness for CQs. Results for the instance-specific subproblem follows from p-correctness. For the instance-independent subproblem, we show that pd-correctness for full CQs can be determined in polynomial time (while it remains NP-hard for general CQs).

2 Preliminaries

Basics. We adopt the named definition of a relation: a relation is of the form $R(A_1, \ldots, A_r)$. Here, R is the relation name, and $A_1, \ldots, A_r$ are the attributes of the relation; we assume that the attributes are disjoint across different relations. We say that $\mathsf{ar}(R) = r$ is the *arity* of the relation. We denote the attribute set of a relation R by $\mathsf{att}(R)$. We also associate with each relation R a collection of *functional dependencies*, which we denote by $\mathsf{fd}(R)$. Given a subset of attributes $\mathbf{A} \subseteq \mathsf{att}(R)$, we denote by $\mathbf{A}^+$ the *fd closure* of $\mathbf{A}$ w.r.t. the functional dependencies in $\mathsf{fd}(R)$. A *database schema* Σ is a finite collection of relations.

We assume a (possibly infinite) domain $\mathbf{dom}$. An *instance* of a relation R is a finite set of tuples of the form $R(a_1, \ldots, a_r)$, where $a_i \in \mathbf{dom}$, and r is the arity of relation R. Given a tuple $t = R(a_1, \ldots, a_r)$, and an attribute A_i, we write $t[A_i]$ to denote the value of t at the position A_i, i.e. $t[A_i] = a_i$. We naturally extend this notation to $t[\mathbf{A}]$, where $\mathbf{A} \subseteq \mathsf{att}(R)$. A *database instance* I over a schema Σ is a collection of relation instances R^I for each relation R in the schema Σ.

Join Condition. Given two relations R, S, we define a *join condition* λ between R, S to be a symmetric binary relation over $\mathsf{att}(R) \cup \mathsf{att}(S)$. Whenever $(A, B), (B, A) \in \lambda$, we will simply write $A = B$. A join condition corresponds to an equi-join between R, S: for example, the join condition $\{A_1 = B_1, A_2 = B_2\}$ describes the equi-join $R \bowtie_{A_1=B_1 \wedge A_2=B_2} S$. This formalization allows a join condition to contain equality on predicates that belong in the same relation. Given a binary relation λ, we denote by $\lambda^{\oplus}$ the minimum equivalence relation that contains λ.

▶ **Example 3.** Consider two relations $R(A, B), S(C, D)$ and the join condition $\lambda = \{A = C, B = C\}$. Then, the equivalence relation $\lambda^{\oplus}$ is $\{A = A, B = B, C = C, A = C, B = C, A = B\}$.

Conjunctive Queries. A CQ is an expression of the form $q(\mathbf{y}) = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell)$. The tuples y and $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ consist of variables and/or constants. Here, $q(\mathbf{y})$ is called the head, $R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell)$ are called atoms and form the body. The symbols $\mathbf{y}$, $\mathbf{x}_1 \ldots \mathbf{x}_\ell$ are vectors that may contain variables or constants. The variables in the head must be a subset of the variables that appear in the body. A CQ is *full* if every variable in the body appears in the head as well, and it is *boolean* if the head contains no variables, i.e., it is of the form $Q()$. A valuation v is a mapping from the variables in q to the constants in $\mathbf{dom}$. We extend v to be the identity mapping for constants. A valuation v satisfies q on instance I if for every $i = 1, \ldots, \ell$, $v(\mathbf{x}_i) \in I$. We define the output $q(I)$ to be the set of all $v(\mathbf{y})$, for a valuation v that satisfies q on I.

Given a CQ q, a *fractional edge packing* is an assignment of a non-negative weight w_i to each atom R_i such that for each variable x_i in q, the sum of the weights of the atoms that contain x_i is at most 1. The fractional edge packing number $\tau^*(q)$ is the maximum value of the quantity $\sum_i w_i$ over all possible fractional edge packings of q.

3 Data Distribution Schemes

Let I be a database instance over a schema Σ. Given a set of p nodes, $\mathcal{N} = \{1, 2, \ldots, p\}$, our goal is to distribute the tuples of I over these p nodes, such that each node $i \in \mathcal{N}$ receives a subset $I_i \subseteq I$.

▶ **Definition 4** (Distribution Scheme). *Let $\mathcal{N} = \{1, 2, \ldots, p\}$ be a set of nodes, and Σ a schema. A* distribution scheme $\mathbf{P}$ *is a function that takes as input an instance I over the schema Σ and a tuple $t \in I$, and returns a set $\mathbf{P}(t, I) \subseteq \mathcal{N}$.*

Given a distribution scheme $\mathbf{P}$, we define the *data chunk* for node i as $\mathbf{P}^{(i)}(I) = \{t \in I \mid i \in \mathbf{P}(t, I)\}$. A distribution scheme can assign a tuple to the empty set, or even replicate a tuple by assigning to multiple nodes. A distribution scheme is deterministic, but in practice we often want to introduce randomness in how the input is distributed. We model this by considering a *family of distribution schemes* $\mathcal{P} = \{\mathbf{P}_1, \mathbf{P}_2, \ldots\}$ defined over the same set of nodes $\mathcal{N}$ and the same schema Σ. Intuitively, to distribute the data we will choose a distribution scheme from the family $\mathcal{P}$ uniformly at random.

3.1 Properties of Distribution Schemes

We next introduce several important properties that a distribution scheme can satisfy.

Obliviousness. A desirable property of a distribution scheme is that the set of nodes where each tuple is assigned depends only on the relation it belongs to and its attribute values, and not the entire instance.

▶ **Definition 5** (Obliviousness). *Let Σ be a schema, and $R \in \Sigma$. We say that R is* oblivious *w.r.t. a family of distribution schemes $\mathcal{P}$ if for every $\mathbf{P} \in \mathcal{P}$ it holds that $\mathbf{P}(t, I) = \mathbf{P}(t, I')$ for every tuple t and pair of instances I, I' over schema Σ such that $t \in R^I, R^{I'}$.*

If every relation of the schema is oblivious w.r.t. $\mathcal{P}$, we say that $\mathcal{P}$ is oblivious, and we can simply express the distribution function as $\mathbf{P}(t)$ (for every $\mathbf{P} \in \mathcal{P}$). Such a scheme is referred to as a *distribution policy* in [3, 4]. A standard example of an oblivious distribution scheme is *hash-partitioning*, where each relation in the schema is hash-partitioned (according to a subset of the attributes) independently of the other relations. Another example is the Hypercube distribution scheme discussed in [3, 15].

An advantage of a distribution policy is that the location of each tuple can be decided by just examining the particular tuple. On the other hand, as we will see later in this section, a distribution policy often limits the way we can distribute data among the nodes, especially when we want to increase the locality of the data in terms of join computation. Recent work [23, 22, 7] has introduced distribution schemes that are not oblivious, and are designed to support join computation locally.

Clustering. A second desirable property of a distribution scheme is that the tuples of a relation R are *clustered* with respect to a set of attributes in R.

▶ **Definition 6** (Clustering). *Let Σ be a schema, $R \in \Sigma$ and $\mathbf{A} \subseteq \mathsf{att}(R)$. We say that R is* $\mathbf{A}$-clustered *w.r.t. a family of distribution schemes $\mathcal{P}$ over a set of nodes $\mathcal{N}$ if for every $\mathbf{P} \in \mathcal{P}$, for every instance I, and any two tuples $t, t' \in R^I$ such that $t[\mathbf{A}] = t'[\mathbf{A}]$, there exists a (unique) node $n \in \mathcal{N}$ such that $\mathbf{P}(t, I) = \mathbf{P}(t', I) = \{n\}$.*

In other words, if R is $\mathbf{A}$-clustered, then all the tuples that have the same values for $\mathbf{A}$ (i.e., are in the same group) always end up in the same node (and only one). In practice, this means that any `group-by` query on R where the grouping attributes are a subset of $\mathbf{A}$ can be computed locally, without any data shuffling. In the special case where $\mathbf{A} = \mathsf{att}(R)$, being $\mathbf{A}$-clustered simply means that for every tuple t, $|\mathbf{P}(t, I)| = 1$, or in other words that there is *no redundancy* in the distribution of the tuples in R. In this case, we will say that R is *non-redundant* w.r.t. to $\mathcal{P}$.

Balancedness. A third desirable property of a distribution scheme is that the storage overhead is small, and the data is partitioned across the nodes in a balanced way. A first attempt to model this would be using the *replication factor*, defined as $r(\mathcal{P}, I) = \max_{\mathbf{P}\in\mathcal{P}} \sum_i |\mathbf{P}^{(i)}(I)|/|I|$. However, the trivial distribution scheme that sends all tuples to a single node has perfect locality and the smallest possible replication factor ($r = 1$). To overcome this problem, we define the notion of balancedness:

▶ **Definition 7** (Balancedness). *Let Σ be a schema, and $\mathcal{N}$ be a set of p nodes. The* balancedness *of a family of distribution schemes $\mathcal{P}$ over Σ and $\mathcal{N}$, and an instance I is*

$$b(\mathcal{P}, I) = \frac{p}{|I|} \cdot \mathbb{E}_{\mathbf{P}\sim\mathcal{P}} \left[\max_{i\in\mathcal{N}} |\mathbf{P}^{(i)}(I)| \right].$$

Here, $\mathbf{P} \sim \mathcal{P}$ means we sample uniformly at random a distribution scheme from $\mathcal{P}$.

Intuitively, the balancedness tells us how much larger the (expected) maximum-sized chunk is compared to $|I|/p$, which is what a perfect splitting of the data would achieve. Note that we can define the balancedness at a relation-level as well. The lemma below gives some intuition about the values that balancedness can take: it is at least as large as the replication factor, but cannot exceed p.

▶ **Lemma 8.** $r(\mathcal{P}, I) \leq b(\mathcal{P}, I) \leq p$

3.2 Queries over Distribution Schemes

Given a distribution of an instance over the nodes, an important question is how a query q can be computed over the given distribution. In this section, we introduce two notions that capture when a distribution scheme is amenable to efficient distributed query computation.

First, we ask whether it is possible to compute a query q by performing exclusively local computation, without any data shuffling. In the case that this is possible, we say that q is *parallel correct*, following the definition in [3].

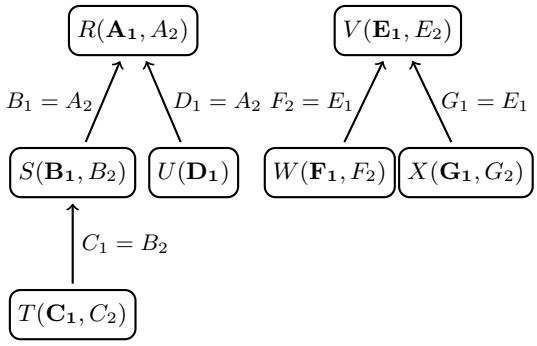
▶ **Definition 9** (Parallel Correctness). *Let $\mathcal{P}$ be a family of distribution schemes over schema Σ and $\mathcal{N} = \{1, \ldots, p\}$ nodes. A query q is* parallel correct (p-correct) *on instance I w.r.t. $\mathcal{P}$ if for every $\mathbf{P} \in \mathcal{P}$, we have $q(I) = \bigcup_{i=1}^{p} q(\mathbf{P}^{(i)}(I))$. A query q is p-correct w.r.t. $\mathcal{P}$ if it is p-correct w.r.t. $\mathcal{P}$ on every instance I over Σ.*

Parallel correctness implies that the query result is the union of the local query results at each node. However, it is possible that a tuple is present in the local results of multiple nodes (i.e., there is redundancy in the result). In this case, getting the correct result would require a deduplication step after the result has been computed. To capture the case where deduplication is not necessary, we need to define a stronger notion of parallel-correctness.

▶ **Definition 10** (Parallel Disjoint Correctness). *Let $\mathcal{P}$ be a family of distribution schemes over schema Σ and $\mathcal{N} = \{1, \ldots, p\}$ nodes. A query q is* parallel disjoint correct (pd-correct) *w.r.t. $\mathcal{P}$ if for every $\mathbf{P} \in \mathcal{P}$ and for every instance I over Σ, $\{q(\mathbf{P}^{(i)}(I))\}_{i\in\mathcal{N}}$ is a partition of the query result $q(I)$.*

3.3 Limitations of Oblivious Schemes

An ideal distribution scheme should be easy to compute (oblivious), cheap to store (balancedness), and also able to minimize the amount of data shuffling in order to compute a given query (parallel-correctness). With the next result, we show that there is a fundamental limitation on what an oblivious distribution scheme can achieve. This result is a straightforward corollary of existing lower bounds on parallel query evaluation [6].



(a) Co-hash graph.

R

A_1	A_2
1	2
1	3
2	3
3	4

S

B_1	B_2
2	4
2	5
3	5
8	9

T

C_1	C_2
4	8
5	7
5	8

U

D_1
2
3
8

V

E_1	E_2
1	5
2	6
8	7

W

F_1	F_2
2	1
1	2
8	3

X

G_1	G_2
2	8
3	1
8	6

(b) Database instance.

R

A_1	A_2
1	2
1	3
2	3
3	4

S

B_1	B_2
2	4
2	5
3	5
3	5
8	9

T

C_1	C_2
4	8
5	7
5	7
5	8
5	8

U

D_1
2
3
3
8

V

E_1	E_2
1	5
2	6
8	7

W

F_1	F_2
2	1
1	2
8	3

X

G_1	G_2
2	8
3	1
8	6

(c) Induced data distribution.

Figure 1 Depiction of a co-hash graph and the resulting data distribution. Figure (c) shows the distribution of tuples induced by each of the co-hash trees. Each color corresponds to a different node.

▶ **Theorem 1.** *Let $\mathcal{P}$ be an oblivious family of distribution schemes over schema Σ and nodes $\mathcal{N} = \{1, \ldots, p\}$. If a self-join-free join query q is parallel-correct w.r.t. $\mathcal{P}$, then for every instance I, we have $b(\mathcal{P}, I) = \Omega(p^{1-1/\tau^*(q)})$, where $\tau^*(q)$ is the fractional edge packing number of q.*

It is important to note that the bound holds for *any instance* I, even if the given instance is easy to distribute efficiently. We should also remark that Theorem 1 holds even for distribution schemes where a tuple $t \in R^I$ is distributed by taking all of R^I into account (but not any other relation). Hence, to overcome the lower bound from Theorem 1, the distribution scheme has to use information from other relations in the instance.

▶ **Example 11.** Consider the conjunctive query $q(x, y, z, w) = R(x, y), S(y, z), T(z, w)$. The maximum fractional edge packing for q is $\tau^* = 2$, hence Theorem 2 implies that any oblivious scheme such that q is p-correct must have balancedness $\Omega(p^{1/2})$ for any instance I. We will see in the next section how locality-aware distribution schemes can overcome this bound.

4 Co-Hashing

We now introduce a class of non-oblivious distribution schemes, which we call *co-hash schemes*, that have been widely used in several practical settings [23, 22, 7].

4.1 Formal Framework

We start by recalling the definition of a rooted in-tree. A *rooted in-tree* is a directed tree such that (i) a single designated vertex is called the *root*, and (ii) every other vertex points *towards* the root.

▶ **Definition 12** (Co-Hash Graph). *Let Σ be a schema. A* co-hash graph $\mathbb{G} = (V, E, \alpha, \lambda)$ *for Σ is an edge-labelled and vertex-labelled directed graph where:*

- *the vertex set V consists of relations in Σ;*
- *the label of a vertex R, denoted $\alpha(R)$, is a vector of attributes from $\mathsf{att}(R)$;*
- *the label of an edge $e = (R, S)$, denoted $\lambda(e)$, is a join condition between R, S; and*
- *the edge set E consists of the union of the disjoint sets of edges of the rooted in-trees.*

Each rooted in-tree in $\mathbb{G}$ is called a *co-hash tree*. The *root path* of a vertex R is the unique directed path from R to the root of its co-hash tree. A co-hash graph does not need to include as vertices all relations from the schema. We will refer to the vertex label $\alpha(R)$ as the *hash signature* of R, and denote the attribute at the i-th position as $\alpha(R)_i$.

▶ **Example 13.** Consider the schema: $\Sigma = \{R(A_1, A_2), S(B_1, B_2), T(C_1, C_2), U(D_1)\}$, and a co-hash tree with four vertices R, S, T, U. Let the vertex labels be: $\alpha(R) = \langle A_1 \rangle, \quad \alpha(S) = \langle B_1 \rangle, \quad \alpha(T) = \langle C_1 \rangle, \quad \alpha(U) = \langle D_1 \rangle$.

There are three edges: $e_{SR} = (S, R)$ with join condition $\lambda(e_{SR}) = \{B_1 = A_2\}$, $e_{TS} = (T, S)$, with join condition $\lambda(e_{TS}) = \{C_1 = B_2\}$, and $e_{UR} = (U, R)$ with join condition $\lambda(e_{UR}) = \{D_1 = A_1\}$. Figure 1a depicts the above co-hash tree.

Induced Distribution. We now explain how a co-hash graph $\mathbb{G}$ induces a family of distribution schemes $\mathcal{P}_\mathbb{G}$ over a set of nodes $\mathcal{N}$. Let I be an instance of the schema Σ. For a tuple $t \in R$, we define its *upwards join set* $\mathcal{J}^\mathbb{G}_\uparrow(t, I)$ in $\mathbb{G}$ as follows. If R is a root of some co-hash tree, then $\mathcal{J}^\mathbb{G}_\uparrow(t, I) = \emptyset$. If R is not the root, then let S be its unique parent in the co-hash tree. Then, $\mathcal{J}^\mathbb{G}_\uparrow(t, I)$ is the set of all tuples from S that join with t on the join condition $\lambda((R, S))$.

For the following, we will assume a family of hash functions $h = \{h^{(1)}, h^{(2)}, \dots\}$, where the hash function $h^{(i)}$ takes i attribute values as input and returns a value from $\mathcal{N}$; we will simply use the notation h when the context is clear.

We can now define recursively the distribution scheme $\mathbf{P}^h_\mathbb{G}$ of the instance I as specified by the co-hash tree $\mathbb{G}$ and the hash family h. For every tuple $t \in R^I$ we define:

$$\mathbf{P}^h_\mathbb{G}(t, I) := \begin{cases} h(t[\alpha(R)_1], t[\alpha(R)_2], \dots), & \text{if } \mathcal{J}^\mathbb{G}_\uparrow(t, I) = \emptyset \\ \bigcup_{s \in \mathcal{J}^\mathbb{G}_\uparrow(t,I)} \mathbf{P}^h_\mathbb{G}(s, I), & \text{otherwise.} \end{cases}$$

In other words, if a tuple t has an empty upwards join set, then it is hash-partitioned using the attributes in $\alpha(R)$. For simplicity of notation, we will often write $\alpha(t)$ to denote the vector $\langle t[\alpha(R)_1], t[\alpha(R)_2], \dots \rangle$. If the upwards join set is not empty, then it is collocated with every tuple in $\mathcal{J}^\mathbb{G}_\uparrow(t, I)$. The data distribution scheme is always well-defined, since we require that the co-hash graph is a collection of disjoint rooted in-trees.

Given a co-hash graph $\mathbb{G}$, we denote by $\mathcal{P}_\mathbb{G} = \{\mathbf{P}^h_\mathbb{G}\}_h$ the family of all distribution schemes $\mathbf{P}^h_\mathbb{G}$, parameterized by all possible hash functions.

▶ **Example 14.** Figure 1c depicts the induced data distribution on our example instance.

Since R is the root of the co-hash tree, the tuples of R will be hashed using attribute $\langle A_1 \rangle$. For example, $R(1, 2), R(1, 3)$ will always end up in the same node (color red). The tuples from S will be co-hashed according to the join condition $R.A_2 = S.B_1$. For instance, the tuple $S(3, 5)$ joins with two tuples from R, $R(1, 3)$ and $R(2, 3)$ and thus will be assigned to where $R(1, 3)$ is (red), and where $R(2, 3)$ is (teal). On the other hand, the tuple $S(8, 9)$ does not join with any tuple from R, and hence it will be hashed on attribute $\langle B_1 \rangle$ (magenta).

In the case where $\mathbb{G}$ has no edges, the resulting data distribution reduces to a hash-partitioning strategy, where each relation is distributed independently according to a subset of its attributes.

4.2 Practical Considerations

Our definition of a co-hash distribution scheme is based on the concept of *predicate-based reference partitioning* [23]. In the most general version, the join condition $\lambda(e)$ can be any predicate. However, as in [23], we restrict our study to equi-joins, since (i) they cover almost all practical scenarios, (ii) other join conditions (e.g., disequality, inequality) may lead to very large replication. We should note here a fundamental difference with [23]. If a tuple in a non-root relation does not join with any tuple from its parent relation, we make sure that the tuple is hashed. In predicate-based reference partitioning, such a tuple is instead arbitrarily distributed to a node. However, this can hurt data locality, as the next example shows.

▶ **Example 15.** Consider the query $q(x, y) = S(x, y), U(x)$. As we will see in Section 6 q is p-correct for the co-hash graph in Figure 1a. On the other-hand, if we do not specify explicitly that the non-joining tuples of R, U are hashed according to B_1, D_1 respectively, the query would not be p-correct. To see this, consider the instance $\{S(a, b), U(a), R(c, d)\}$. Note that the tuples $S(a, b), U(a)$ satisfy q, but since neither of the two joins with R, they will be assigned to arbitrary nodes.

The declarative framework of distribution constraints introduced in [11] also captures predicate-based reference partitioning, but it cannot control how the non-joining tuples are assigned to nodes as we do in this paper. As we show in the above example, this limitation means that fewer queries may be p-correct for a given scheme.

4.3 Some Useful Notions

We next discuss notions and properties of co-hash graphs that will be used throughout the paper.

▶ **Definition 16** (Terminating Path). *Let $\mathbb{G}$ be a co-hash graph, and I an instance. The tuple sequence $t_0 \to t_1 \to \cdots \to t_k$ is a* terminating path *for $t_0 \in I$ if (i) for every $j = 0, \ldots, k-1$ we have $t_{j+1} \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(t_j, I)$, and (ii) $\mathcal{J}^{\mathbb{G}}_{\uparrow}(t_k, I) = \emptyset$.*

Note that the above terminating path for t_0 implies that t_0 will be assigned to node $h(\alpha(t_k))$ for the scheme $\mathbf{P}^h_{\mathbb{G}}$.

We also define a *hash destination* to be a vector of values that is passed to the hash function h. For a tuple $t \in I$, $H(t, I)$ is the set of hash destinations that is used to assign tuple t to a node. For instance, if $t \in R^I$ and R is a root node in $\mathbb{G}$ with $\alpha(R) = \langle A, B\rangle$, then $H(t, I) = \{\langle t[A], t[B]\rangle\}$.

Tuple Collocation. Given a co-hash graph $\mathbb{G}$, an instance I and a set of tuples $S \subseteq I$, we write $I \rhd_{\mathbb{G}} S$ if for every $\mathbf{P} \in \mathcal{P}_{\mathbb{G}}$ we have $\bigcap_{s \in S} \mathbf{P}(s, I) \neq \emptyset$. In other words, $I \rhd_{\mathbb{G}} S$ if the tuples from S are always collocated in some common node, no matter the choice of the hash family. We show next that we can decide this problem in P. Intuitively, this holds because of the acyclic structure of the co-hash graph.

▶ **Lemma 17.** *Let $\mathbb{G} = (V, E, \alpha, \lambda)$ be a co-hash graph, I be an instance, and $S \subseteq I$. Then, we can decide in polynomial time whether $I \rhd_{\mathbb{G}} S$.*

5 Properties of Co-hashing

5.1 Balancedness

We first discuss how co-hash schemes can overcome the lower bound for oblivious schemes.

▶ **Example 18.** Consider again the query we used in Example 11, $q(x, y, z, w) = R(x, y), S(y, z), T(z, w)$. Suppose that the relations R, S, T are distributed according to the co-hash graph of Figure 1a. It is easy to see that q is parallel-correct.

We now turn into analyzing the balancedness of the distribution scheme induced by the co-hash graph. Suppose that A_1 is a key in R, B_1 is a key in S, and C_1 is a key in T; this is a common scenario, since many joins in practice are key-foreign key joins. In this case, assuming that $|I| \gg p$, one can show that the balancedness will be $b = O(1)$, which is asymptotically close to the best possible value of 1. On the other hand, as we showed in the previous section, any oblivious scheme will have balancedness $\Omega(p^{1/2})$.

Our first result generalizes the above example. We show that if the hash signatures and join conditions involve keys of the relations, balancedness is guaranteed to be constant. This result captures a lot of real-world examples, since in practice co-hash graphs are constructed by following the key-foreign key constraints of the schema.

▶ **Lemma 19.** *Let $\mathbb{G}$ be a co-hash graph such that (i) for every edge $e = (R, S)$, the attributes in $\lambda(e)$ form a superkey for R, and (ii) for every vertex R, the attributes in $\alpha(R)$ form a superkey for R. Let I be an instance such that $|I| \gg p$. Then, $b(\mathcal{P}_{\mathbb{G}}, I) = O(1)$.*

Our second result shows that any (non-trivial) co-hash partitioning scheme has constant balancedness when the input database has bounded degree. Formally, a *bounded degree* database is one where the number of times each value appears in a tuple is bounded by a constant $d \in \mathbb{N}$. This is not a surprising result, since the constraint of bounded degree means that each tuple can join with at most d other tuples. Contrast this result with oblivious schemes, where even for bounded degree instances the balancedness is a non-constant function of the number of nodes p.

▶ **Lemma 20.** *Let $\mathbb{G}$ be a co-hash graph such that (i) no join conditions are empty, and (ii) every hash signature is of size at least 1. Let I be a bounded degree instance with bound d such that $|I| \gg p$. Then $b(\mathcal{P}_{\mathbb{G}}, I) = O(d^{\ell})$, where ℓ is the maximum height of a rooted in-tree in $\mathbb{G}$.*

5.2 Obliviousness

Let Σ be a database schema and $\mathbb{G}$ a co-hash graph over Σ. The question we ask here is: which relations in Σ are oblivious w.r.t. $\mathcal{P}_{\mathbb{G}}$?

CoHash-Oblivious	
Input:	co-hash graph $\mathbb{G}$, relation R
Question:	Is R oblivious with respect to $\mathcal{P}_{\mathbb{G}}$?

Before we describe how we can decide the above property, we need some additional technical machinery.

▶ **Definition 21** (Hash-Compatible). *Let $\mathbb{G} = (V, E, \alpha, \lambda)$ be a co-hash graph, and $R, S \in V$. We say that R is* hash-compatible *with S w.r.t. an equivalence relation ρ over* att*, denoted $\rho \models R \parallel S$, if:*

1. *$\alpha(R), \alpha(S)$ have the same arity; and*
2. *for every position i, $\alpha(R)_i =_\rho \alpha(S)_i$.*

▶ **Example 22.** Consider the co-hash tree rooted at R in the co-hash graph presented in Figure 1a. Consider relations R and S, and let $\rho = \{A_1 = B_1, A_2 = B_1\}$.

We claim that $\rho^\oplus \models R \parallel S$. Indeed, $|\alpha(R)| = |\alpha(S)| = 1$, so the arities of R and S are the same. Also, $\alpha(R)_1 = A_1$, $\alpha(S)_1 = B_1$, and $A_1 = B_1 \in \rho^\oplus$. Hence, $\alpha(R)_1 =_{\rho^\oplus} \alpha(S)_1$.

On the other hand, if $\rho = \{A_2 = B_1\}$, then $\alpha(R)_1 \neq_{\rho^\oplus} \alpha(S)_1$, and $\rho^\oplus \models R \parallel S$ does not hold.

▶ **Lemma 23.** *Let $\mathbb{G} = (V, E, \alpha, \lambda)$ be a co-hash graph, and $R \in V$. Let $(R =)S_0 \xrightarrow{e_1} S_1 \xrightarrow{e_2} \ldots \xrightarrow{e_k} S_k$ be the (unique) root path of R. If $(\lambda(e_1) \cup \cdots \cup \lambda(e_j))^\oplus \models R \parallel S_j$ for every node $j = 1, \ldots, k$, then for every hash family h and every instance I we have $\mathbf{P}^h_\mathbb{G}(t, I) = \{h(\alpha(t))\}$.*

We can now state the main theorem of this section.

▶ **Theorem 2.** *Let $\mathbb{G} = (V, E, \alpha, \lambda)$ be a co-hash graph, and $R \in V$. Let $(R =)S_0 \xrightarrow{e_1} S_1 \xrightarrow{e_2} \ldots \xrightarrow{e_k} S_k$ be the root path of R. The following are equivalent:*

1. *$\mathcal{P}_\mathbb{G}$ is oblivious for R.*
2. *R is hashed on $\alpha(R)$.*
3. *For every $i = 1, \ldots, k$, we have $\rho_i \models R \parallel S_i$, where $\rho_i = (\lambda(e_1) \cup \cdots \cup \lambda(e_i))^\oplus$.*

▶ **Example 24.** Consider the co-hash tree rooted at V in the co-hash graph presented in Figure 1a. Consider relation X in the co-hash tree. The root path of X is $X \to V$. We have $\rho_1 = \{G_1 = E_1\}$, which means that $\rho_1 \models R \parallel V$ holds. Hence, condition (3) of Theorem 2 holds and X is oblivious.

▶ **Proposition 1.** *CoHash-Oblivious is in* P.

5.3 Clustering

In this section we consider whether a co-hash graph $\mathbb{G}$ induces a clustering of a relation in the schema. Recall that by checking whether a relation is clustered, we implicitly also check about the presence of redundancy.

CoHash-Clustered	
Input:	co-hash graph $\mathbb{G}$, relation R, $\mathbf{A} \subseteq \mathsf{att}(R)$
Question:	Is R $\mathbf{A}$-clustered w.r.t. $\mathcal{P}_\mathbb{G}$?

Algorithm 1 Checking Clustering.

Input: $\mathbb{G} = (V, E, \alpha, \lambda)$, $R \in V$, $\mathbf{A} \subseteq \mathsf{att}(R)$
Let root path $(R =)S_0 \xrightarrow{e_1} S_1 \xrightarrow{e_2} \ldots \xrightarrow{e_k} S_k$
$\mathcal{V}_0 \leftarrow \mathbf{A}^+$
$i \leftarrow 0$
while $i < k$ **do**
  if $\exists j$ s.t. $\alpha(S_i)_j \notin \mathcal{V}_i$ **then** ▷ (VC)
    return false
  if $\exists (A, B) \in \lambda(e_{i+1})$ s.t. $A \notin \mathcal{V}_i$ **then** ▷ (EC)
    break
  i++
  $\mathcal{V}_i \leftarrow \{B \in \mathsf{att}(S_i) \mid \exists A \in \mathcal{V}_{i-1} : (A = B) \in \lambda(e_i)\}^+$
if $i = k$ and $\exists j : \alpha(S_k)_j \notin \mathcal{V}_k$ **then**
  return false
else
  for $j = i + 1, \ldots, k$ **do**
    if $(\lambda(e_1) \cup \ldots \cup \lambda(e_j))^{\oplus} \models S_i \parallel S_j$ **then** ▷ (HC)
      return false
return true ;

An easy observation is that obliviousness implies that R is $\mathsf{att}(R)$-clustered, and so it is non-redundant. Indeed, from Theorem 2, an oblivious relation allocates each tuple to exactly one location, and thus it is always non-redundant. The inverse is not true, as the next example demonstrates.

▶ **Example 25.** Consider two relations, $R(A_1, A_2)$, $S(B_1, B_2)$, and a co-hash graph $\mathbb{G} = (V, E, \alpha, \lambda)$ with $V = \{R, S\}$, $E = \{(S, R)\}$ and $\lambda((S, R)) = \{B_1 = A_1\}$. Moreover, let $\alpha(S) = \langle B_2 \rangle$ and $\alpha(R) = \langle A_1 \rangle$.

Observe first that S is not oblivious, since S is not hash-compatible with R w.r.t. $\{B_1 = A_1\}^{\oplus}$. We will argue next that S is non-redundant. Consider any instance I and a tuple $s \in S^I$. If s does not join with any tuple in R^I on $A_1 = B_1$, then it is simply hashed on $\langle B_2 \rangle$ and is non-redundant. So suppose that s joins with tuples $\{r_1, \ldots, r_n\}$ in R^I. Hence, s will end up in the nodes $\{h(r_1[A_1]), \ldots, h(r_n[A_1])\}$. However, for any two tuples r_i, r_j, we have $r_i[A_1] = s[B_1] = r_j[A_1]$, and hence $h(r_1[A_1]) = h(r_2[A_1]) = \cdots = h(r_n[A_1])$, which means that s will be assigned to exactly one node.

A second observation is that functional dependencies are now critical in deciding whether there is redundancy (and in general clustering) in a relation or not.

▶ **Example 26.** Consider the same two relations $R(A_1, A_2)$ and $S(B_1, B_2)$, and a co-hash graph $\mathbb{G} = (V, E, \alpha, \lambda)$ with $V = \{R, S\}$, $E = \{(S, R)\}$, $\lambda((S, R)) = \{B_1 = A_1\}$. Moreover, let $\alpha(S) = \langle B_2 \rangle$ and $\alpha(R) = \langle A_2 \rangle$.

In contrast to the previous example, where the hash signature of R was $\langle A_1 \rangle$, S is now redundant. However, if we add to relation R the functional dependency $A_1 \to A_2$, S becomes non-redundant.

We now present an algorithm (Algorithm 1) that decides in polynomial time whether a relation R is $\mathbf{A}$-clustered for some $\mathbf{A} \subseteq \mathsf{att}(R)$ (and hence also decides non-redundancy).

The algorithm takes as input the co-hash graph $\mathbb{G}$, the relation R, and an attribute set $\mathbf{A} \subseteq \mathsf{att}(R)$. The algorithm starts from the node R, and then traverses the root path of R bottom up. At each relation S_i of the root path, it computes inductively a set of attributes

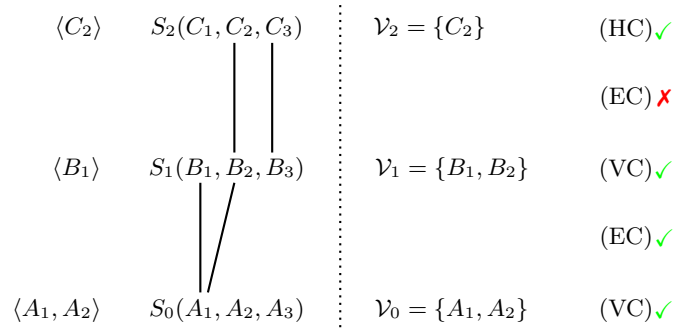


Figure 2 Example execution of Algorithm 1 to check whether relation S_0 is $\mathbf{A}$-clustered, where $\mathbf{A} = \{A_1, A_2\}$. The edges between attributes depict the join conditions.

$\mathcal{V}_i \subseteq \mathsf{att}(S_i)$. While computing each $\mathcal{V}_i$, we also take into account functional dependencies. It then uses this set to check two conditions: one condition (EC) on each edge of the root path, and one condition (VC) on each vertex of the root path. If the vertex condition fails, the algorithm returns false. Otherwise, if the edge condition (EC) is not satisfied, then we check the remaining path for hash compatibility (HC).

We illustrate the working of Algorithm 1 in an example:

▶ **Example 27.** Consider the co-hash graph depicted in Figure 2. Suppose there are no functional dependencies between attributes in any of the relations. We will run the algorithm to check whether S_0 is $\mathbf{A}$-clustered, where $\mathbf{A} = \{A_1, A_2\}$. We initially have $\mathcal{V}_0 = \mathbf{A}^+ = \{A_1, A_2\}$. The first vertex condition (VC) for $i = 0$ is satisfied, since both attributes in the hash signature of S_0 are in $\mathcal{V}_0$. The first edge condition (EC) is also satisfied, since the join condition between S_0, S_1 only uses attribute A_1, which belongs in $\mathcal{V}_0$. Similarly, one can check that the second (VC) is also satisfied. However, the second (EC) for $i = 1$ fails, since $B_3 = C_3 \in \lambda(e_2)$, but $B_3 \notin \mathcal{V}_1$. The algorithm now enters the loop in lines 15-17, where it checks for hash compatibility between S_1 and all the relations up to the root. In this example, we need to check only one (HC), between S_1 and S_2. Since $B_1 = C_2 \in (\lambda(e_1) \cup \lambda(e_2))^{\oplus}$, hash compatibility holds, and thus the algorithm returns true. Thus, S_0 is $\{A_1, A_2\}$-clustered. This also implies that S_0 is $\{A_1, A_2, A_3\}$-clustered, so also non-redundant.

▶ **Theorem 3.** *Let $\mathbb{G} = (V, E, \alpha, \lambda)$ be a co-hash graph, relation $R \in V$, and $\mathbf{A} \subseteq \mathsf{att}(R)$. Algorithm 1 runs in polynomial time (in the size of $\mathbb{G}$) and returns true if and only if R is $\mathbf{A}$-clustered w.r.t. $\mathcal{P}_{\mathbb{G}}$. Hence, CoHash-Clustered is in* P.

6 Parallel Correctness

In this section, we investigate the problem of p-correctness for data distributions induced by a co-hash graph. We start by considering the following decision problem, where we check for p-correctness for an instance that is given as input to the problem.

I-CPC	
Input:	co-hash graph $\mathbb{G}$, input I, query q
Question:	Is q p-correct on I under $\mathcal{P}_{\mathbb{G}}$?

For CQs, the above problem is Π_2^P-complete. Interestingly, the hardness result comes from the observation that a co-hash scheme can implicitly encode an arbitrary oblivious distribution scheme. More precisely, for each relation R, the scheme is encoded as another

relation R' that stores the node destinations for each tuple in R. However, we can obtain a better complexity result if we restrict to full CQs, which correspond to joins without projections.

▶ **Theorem 4.** *I-CPC is* Π_2^P*-complete for CQs.*

▶ **Theorem 5.** *I-CPC is* coNP*-complete for full CQs.*

We next consider the problem of p-correctness across all possible instances.

CPC	
Input:	co-hash graph $\mathbb{G}$, query q
Question:	Is q p-correct under $\mathcal{P}_{\mathbb{G}}$?

The next example shows that deciding p-correctness is not always straightforward.

▶ **Example 28.** Consider the co-hash graph in Figure 1a, and the queries $q_1(x, y, z) = R(x, y), S(y, z)$ and $q_2(x, y, z, w) = R(x, y), S(y, z), T(z, w)$. Both queries can be computed locally without any data shuffling, since by construction of the co-hash distribution, tuples that join are placed in the same node. Hence, both are p-correct. However, there are other queries that are p-correct in more indirect ways. For instance, the query $q_3(x, y) = S(x, y), U(x)$ is also p-correct.

Our first result is that the above problem is NP-hard for general CQs. The hardness comes from a reduction from the problem of query containment. It is worth noting that the lower bound holds even for co-hash graphs with no edges, which correspond to oblivious distribution schemes. On the other hand, for full CQs the decision problem for p-correctness is in polynomial time.

▶ **Theorem 6.** *CPC is* NP*-hard for CQs.*

▶ **Theorem 7.** *CPC is in* P *for full CQs.*

Our polynomial time algorithm generates a polynomial number of small instances with labelled nulls (denoted by $\perp_o$) and constants, and then makes a collocation check for each one of the generated instances.

Functional Dependencies. Each node R in the co-hash graph is associated with a set of functional dependencies $\mathsf{fd}(R)$. Let $F = \bigcup_R \mathsf{fd}(R)$. We will need to apply the chase algorithm to an instance I w.r.t. F, which results in a new instance I' (or the chase fails). It will be convenient to capture the result of the chase by a homomorphism θ such that $I' = \theta(I)$. We then write $\theta = \mathit{chase}(I, F)$.

The Extension Step. We now describe a procedure, called $\mathit{extend}(I, t)$, that takes as input an instance I (with labelled nulls and constants) and a tuple $t \in R^I$. The procedure can fail, in which case it returns $\perp$.

If R has no outgoing edge in $\mathbb{G}$, the procedure fails. Otherwise, let $e = (R, S)$ be the unique outgoing edge from R. Intuitively, we will try to extend I with a fresh most general tuple from S that joins with t on $\lambda(e)$.

As a first step, we construct a homomorphism ζ that maps the labelled nulls of I onto themselves (or constants) such that $\zeta(t[A_i]) = \zeta(t[A_j])$ whenever $A_i = A_j \in \lambda(e)^{\oplus}$ with $i \neq j$, otherwise it is the identity mapping. This step may fail when $t[A_i], t[A_j]$ are distinct constants, in which case the procedure fails. Let $I' = \zeta(I)$ and $t' = \zeta(t)$.

As a second step, we construct a new tuple s from S such that for every $A = B \in \lambda(e)$ with $A \in \mathsf{att}(R), B \in \mathsf{att}(S)$ we have $s[B] = t'[A]$, and for every non-joining attribute we introduce a fresh labelled null that does not occur in I'. Because of how we constructed I', the tuple s is always well-defined. Finally, we apply the chase to the instance $I' \cup \{s\}$. If the chase fails, we return $\bot$. Otherwise, let $\chi = chase(I' \cup \{s\}, F)$. The procedure returns the pair $(\chi \circ \zeta, \chi(s))$.

Algorithm. Let $D[q]$ be the canonical instance of q (we assume w.l.o.g. that we have chased q w.r.t. F). If the tuples in $D[q]$ are not collocated, then $D[q]$ is a witness instance that proves that q is not p-correct. Recall that we can do this check in polynomial time using Lemma 17. However, if the tuples are collocated, then this is not a sufficient condition for p-correctness, and we need to check additional instances. We do this through Algorithm 2. It starts with the canonical instance $D[q]$ of the query q, and checks whether the tuples in $D[q]$ are collocated for this instance (Line 2). Then, for every tuple in $D[q]$, it initiates a sequence of extension steps: each step is applied using the tuple generated in the previous step (if possible). The algorithm checks whether the tuples in $D[q]$ are collocated in every instance generated in this fashion; if so, it returns true, otherwise it terminates with false.

Algorithm 2 Deciding P-Correctness.

```
1:  Input: co-hash graph 𝔾, full CQ q
2:  if D[q] ⊭_𝔾 D[q] then
3:      return false ;
4:  for each t̃ ∈ D[q] do
5:      i ← 0 ; D_0 ← D[q] ; s̃_0 ← t̃ ; I_0 ← D[q]
6:      while extend(I_i, s̃_i) ≠ ⊥ do
7:          i ← i + 1 ;
8:          (φ_i, s̃_i) ← extend(I_{i−1}, s̃_{i−1}) ;
9:          I_i ← φ_i(I_{i−1}) ∪ {s̃_i} ;
10:         D_i ← φ_i(D_{i−1}) ;
11:         if I_i ⊭_𝔾 D_i then
12:             return false ;
13: return true ;
```

We analyze the runtime and correctness of the above algorithm in the appendix. Here, we present two examples of its execution.

▶ **Example 29.** Consider the co-hash graph in Figure 1a, and the query $q_4(x, y, z) = R(x, y), S(z, x)$. The canonical instance for this query is $I_0 = \{R(\bot_x, \bot_y), S(\bot_z, \bot_x)\}$. The hash destination for $R(\bot_x, \bot_y)$ is $\langle \bot_x \rangle$, and for $S(\bot_z, \bot_x)$ it is $\langle \bot_z \rangle$, hence the two tuples are not collocated. This means that the algorithm exits early and outputs false.

▶ **Example 30.** Consider the co-hash graph in Figure 1a, and the query $q_3(x, y) = S(x, y), U(x)$. The canonical instance for this query is $I_0 = \{S(\bot_x, \bot_y), U(\bot_x)\}$. The hash destination for $S(\bot_x, \bot_y)$ is $\langle \bot_x \rangle$, and for $U(\bot_x)$ it is $\langle \bot_x \rangle$ as well, hence the two tuples are always collocated in I_0.

The extension step for the tuple $S(\bot_x, \bot_y)$ returns a tuple $R(\bot_w, \bot_x)$ and the identity homomorphism. Hence, $I_1 = I_0 \cup \{R(\bot_w, \bot_x)\}$ and $D_1 = D[q]$. Note that in I_1, both $S(\bot_x, \bot_y)$ and $U(\bot_x)$ join with $R(\bot_w, \bot_x)$, hence the hash destination for both is $\langle \bot_w \rangle$.

Hence, the tuples are collocated in I_1 as well. Since R is a root relation in the co-hash graph, the next extension step fails and the loop terminates.

The extension step for the tuple $U(\perp_x)$ returns a new tuple $R(\perp_v, \perp_x)$ and the identity homomorphism. Hence, $I_1 = I_0 \cup \{R(\perp_v, \perp_x)\}$ and $D_1 = D[q]$. As before, the two tuples from D_1 are collocated. The next extension step fails and thus the loop terminates. The algorithm will now terminate and output that the query is indeed p-correct.

7 Parallel Disjoint Correctness

In this section, we study in analogy to the previous section the problem of pd-correctness.

I-CPDC	
Input:	co-hash graph $\mathbb{G}$, input I, query q
Question:	Is q pd-correct on I under $\mathcal{P}_{\mathbb{G}}$?

CPDC	
Input:	Co-hash graph $\mathbb{G}$, query q
Question:	Is q pd-correct under $\mathcal{P}_{\mathbb{G}}$?

The complexity landscape for pd-correctness follows the same pattern with the corresponding problems for p-correctness.

▶ **Theorem 8.** *I-CPDC is* Π_2^P*-complete for CQs and* coNP*-complete for full CQs.*

▶ **Theorem 9.** *CPDC is* NP*-hard for CQs, and in* P *for the class of full CQs.*

▶ **Example 31.** Consider the co-hash graph in Figure 1a, and the p-correct query $q(x, y, z) = R(x, y), S(z, x)$. The dominant atom, R, is the non-redundant root. This makes q pd-correct.

Consider another p-correct query $q(x, y) = S(x, y), U(x)$ on the same co-hash graph. Both atoms are dominant, and both are redundant. Hence, the query is not pd-correct. However, if we add to relation R the fd $A_2 \to A_1$, then both become non-redundant and q will be pd-correct.

8 Related Work

There has been a lot of work in studying parallel evaluation of queries. The massively parallel communication (MPC) model was introduced in [14, 15] to analyze multiway joins and to obtain bounds on the amount of communication and synchronization [1, 5]. The case of query evaluation in a single round of communication has been of particular interest, where data is shuffled once before a query is evaluated. The notion of parallel correctness was introduced in [3, 4] to study query evaluation in one round w.r.t. a distribution policy. [13] extended the study of parallel correctness of conjunctive queries to incorporate bag semantics and [9] extended the ideas to unions of conjunctive queries with negation. Distribution policies and parallel correctness results have also been studied in the context of Datalog [12, 18].

The studies mentioned above have focused on *oblivious* distribution policies, where the destination of a tuple is independent of the input, whereas our work focuses on locality-aware schemes. Further, we assume that the partitioning step (the data shuffling phase) is done as a preprocessing step before any query is query evaluated. In effect, we study query evaluation with "zero" rounds of communication. This mode of "partition once, run queries multiple times" is the approach taken by multiple systems that are tuned for OLAP style queries [7, 21, 17]. Recently, Geck et. al [11] introduced a declarative framework that captures

constraints of distributed data; these constraints can capture several classes of non-oblivious distribution schemes. The co-hash schemes we describe in this paper cannot be captured by their framework because we explicitly hash tuples that do not join with the parent relation. In addition, we go beyond the instance-independent p-correctness that was examined in [11], and we study several other properties of theoretical and practical interest.

The idea of location-aware partitioning has been deployed in multiple systems. It has been shown to improve query performance drastically for queries that can take advantage of the collocation by reducing data shuffling across nodes [7, 16, 19]. Reference-based partitioning has been proposed in [7] and join-predicate based partitioning in [22, 23]; it is also deployed in [19]. However, none of these proposals study the properties of the resulting partitioning scheme, or how to decide which queries can be executed without any data shuffling. In fact, the query evaluation procedure of [23] ends up shuffling data for queries that could have been evaluated without any data movement.

9 Conclusions and Future Work

In this work, we initiate the formal study of co-hash partitioning, a popular locality-aware data distribution scheme. One immediate direction for future work is to extend the study to UCQs (with negation as well), similar to the extensions in [10]. It is also interesting to consider what happens for queries that are not p-correct for a given co-hash graph. In this case, it will be useful to determine the best data shuffling strategy such that the amount of communication and load per node is minimized. Finally, in this paper we study problems when the co-hash graph is given; an orthogonal problem is to design co-hash schemes that are optimized for a particular query workload.

References

1 Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. Parallel skyline queries. In *ICDT*, pages 274–284, 2012. doi:10.1145/2274576.2274605.

2 Foto N. Afrati and Jeffrey D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, pages 99–110, 2010. doi:10.1145/1739041.1739056.

3 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and transferability for conjunctive queries. In *PODS*, 2015. doi:10.1145/2745754.2745759.

4 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Data partitioning for single-round multi-join evaluation in massively parallel systems. *SIGMOD Record*, 45(1):33–40, 2016. doi:10.1145/2949741.2949750.

5 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *PODS*, pages 273–284, 2013. doi:10.1145/2463664.2465224.

6 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *J. ACM*, 64(6):40:1–40:58, 2017. doi:10.1145/3125644.

7 George Eadon, Eugene Inseok Chong, Shrikanth Shankar, Ananth Raghavan, Jagannathan Srinivasan, and Souripriya Das. Supporting table partitioning by reference in Oracle. In *SIGMOD*, pages 1111–1122, 2008. doi:10.1145/1376616.1376727.

8 Sumit Ganguly, Avi Silberschatz, and Shalom Tsur. Parallel bottom-up processing of Datalog queries. *J. Log. Program.*, 14(1-2):101–126, October 1992. doi:10.1016/0743-1066(92)90048-8.

9 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and containment for conjunctive queries with union and negation. In *ICDT*, pages 9:1–9:17, 2016. doi:10.4230/LIPIcs.ICDT.2016.9.

10 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and containment for conjunctive queries with union and negation. *ACM Trans. Comput. Logic*, 20(3):18:1–18:24, June 2019. doi:10.1145/3329120.

11 Gaetano Geck, Frank Neven, and Thomas Schwentick. Distribution constraints: The chase for distributed data. In *ICDT*, pages 13:1–13:19, 2020. doi:10.4230/LIPIcs.ICDT.2020.13.

12 Bas Ketsman, Aws Albarghouthi, and Paraschos Koutris. Distribution policies for Datalog. In *ICDT*, pages 17:1–17:22, 2018. doi:10.4230/LIPIcs.ICDT.2018.17.

13 Bas Ketsman, Frank Neven, and Brecht Vandevoort. Parallel-correctness and transferability for conjunctive queries under bag semantics. In *ICDT*, pages 18:1–18:16, 2018. doi:10.4230/LIPIcs.ICDT.2018.18.

14 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *PODS*, pages 223–234, 2011. doi:10.1145/1989284.1989310.

15 Paraschos Koutris and Dan Suciu. A guide to formal analysis of join processing in massively parallel systems. *SIGMOD Record*, 45(4):18–27, 2016. doi:10.1145/3092931.3092934.

16 Yi Lu, Anil Shanbhag, Alekh Jindal, and Samuel Madden. AdaptDB: Adaptive partitioning for distributed joins. *Proc. VLDB Endow.*, 10(5):589–600, January 2017. doi:10.14778/3055540.3055551.

17 Rimma Nehme and Nicolas Bruno. Automated partitioning design in parallel database systems. In *SIGMOD*, pages 1137–1148, 2011. doi:10.1145/1989323.1989444.

18 Frank Neven, Thomas Schwentick, Christopher Spinrath, and Brecht Vandevoort. Parallel-correctness and parallel-boundedness for Datalog programs. In *ICDT*, pages 14:1–14:19, 2019. doi:10.4230/LIPIcs.ICDT.2019.14.

19 Jags Ramnarayan, Barzan Mozafari, Sumedh Wale, Sudhir Menon, Neeraj Kumar, Hemant Bhanawat, Soubhik Chakraborty, Yogesh Mahajan, Rishitesh Mishra, and Kishor Bachhav. SnappyData: A hybrid transactional analytical store built on Spark. In *SIGMOD*, pages 2153–2156, 2016. doi:10.1145/2882903.2899408.

20 Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. F1: A distributed SQL database that scales. *Proc. VLDB Endow.*, 6(11):1068–1079, August 2013. doi:10.14778/2536222.2536232.

21 Young-Kyoon Suh, Ahmad Ghazal, Alain Crolotte, and Pekka Kostamaa. A new tool for multi-level partitioning in Teradata. In *CIKM*, pages 2214–2218, 2012. doi:10.1145/2396761.2398604.

22 Khai Q. Tran, Jeffrey F. Naughton, Bruhathi Sundarmurthy, and Dimitris Tsirogiannis. JECB: A join-extension, code-based approach to OLTP data partitioning. In *SIGMOD*, pages 39–50, 2014. doi:10.1145/2588555.2610532.

23 Erfan Zamanian, Carsten Binnig, and Abdallah Salama. Locality-aware partitioning in parallel database systems. In *SIGMOD*, pages 17–30, 2015. doi:10.1145/2723372.2723718.

A Proofs

Proof of Lemma 8. Since $|\mathbf{P}^{(i)}(I)| \leq |I|$ always holds, we have that $b(\mathcal{P}, I) \leq p$. Equality is achieved by the distribution scheme that sends all its data to a single node. For the other direction, for every $\mathbf{P} \in \mathcal{P}$ we have $\sum_i |\mathbf{P}^{(i)}(I)| \geq r(\mathcal{P}, I) \cdot |I|$. Hence, $\max_i |\mathbf{P}^{(i)}(I)| \geq r(\mathcal{P}, I)|I|/p$ and $b(\mathcal{P}, I) \geq r(\mathcal{P}, I)$. ◀

Proof of Theorem 1. The key idea is that if $\mathcal{P}$ is oblivious, then it can be turned into a 1-round algorithm for computing q in the MPC model. In the MPC model, computation is performed in rounds, where each round includes local computation followed by a global exchange of data. Initially, the input data I is distributed arbitrarily across p nodes, and at the end of the computation, the query result must be the union of the local results of each node.

The algorithm works as follows: it chooses $\mathbf{P} \sim \mathcal{P}$ uniformly at random, and then sends each tuple $t \in I$ to the node $\mathbf{P}(t)$. Finally, each node $i \in \mathcal{N}$ computes the query q on its local data, $q(\mathbf{P}^{(i)}(I))$. The correctness of the algorithm follows directly from the fact that q is p-correct w.r.t. $\mathcal{P}$. Additionally, the load of the algorithm L in the MPC model is at least $b(\mathcal{P}, I) \cdot |I|/p$.

We now apply the result from [6], which tells us that any 1-round randomized MPC algorithm on p nodes that computes q correctly must have load $L = \Omega(|I|/p^{1/\tau^*(q)})$. ◀

Proof of Lemma 17. The polynomial time algorithm first computes for every tuple $t \in I$ the set of hash destinations $H(t, I)$. This set is computed inductively starting from the root(s) of the co-hash graph and moving to the leaves. For a tuple t in a root relation, we have that $H(t, I) = \{\alpha(t)\}$, i.e., the set of hash destinations is a singleton. For any non-root relation, we look at the upwards join set of t. If it is not empty, its hash destination is the union of the hash destinations of the tuples in its upwards join set; otherwise, it is $\{\alpha(t)\}$. This process terminates in polynomial time, since (i) the upwards join set can be computed in polynomial time, and (ii) the possible hash destinations are at most $|V| \cdot |I|$, hence polynomially bounded. As a final step, we compute the intersection $\bigcap_{s \in S} H(s, I)$. If the result is nonempty, then for any choice of hash function (and hence any $\mathbf{P} \in \mathcal{P}_{\mathbb{G}}$), the tuples in S will be collocated in some common node. Otherwise, we can always pick a p large enough and a hash function that does not assign all tuples from S to the same node. ◀

Proof of Lemma 19. Let $\langle a_1, \ldots, a_k \rangle$ be a hash destination, where $k \geq 1$. The key observation is that $\langle a_1, \ldots, a_k \rangle$ can be the hash destination of $O(|V|)$ tuples, where V is the vertex set of $\mathbb{G}$. Indeed, by induction it follows that at most one tuple per relation can be assigned to $\langle a_1, \ldots, a_k \rangle$. Since each tuple will be assigned to at least one hash destination, there must be at least $\Omega(|I|/|V|)$ hash destinations. Now, observe that the hash function h sends each hash destination to a node independently and uniformly at random. Let H be the number of hash destinations for instance I. Since $|I| \gg p$, the expected maximum number of hash destinations assigned to each node will be $O(H/p)$. Finally, observe that $H \leq |I|$ and that each hash destination has $O(|V|)$ tuples. Hence, $b(\mathcal{P}_{\mathbb{G}}, I) = \frac{p}{|I|} O(|I||V|/p) = O(|V|)$. ◀

Proof of Lemma 20. Let $\langle a_1, \ldots, a_k \rangle$ be a hash destination, where $k \geq 1$. Our main observation is that $\langle a_1, \ldots, a_k \rangle$ can be the hash destination of $O(d^\ell)$ tuples. Since each tuple will be assigned to at least one hash destination, there must be at least $\Omega(|I|/d^\ell)$ hash destinations. Now, observe that the hash function h sends each hash destination to a node independently and uniformly at random. Let H be the number of hash destinations for instance I. Since $|I| \gg p$, the expected maximum number of hash destinations assigned to each node will be $O(H/p)$. Finally, observe that $H \leq |I|$ and that each hash destination has $O(d^\ell)$ tuples. Hence, $b(\mathcal{P}_{\mathbb{G}}, I) = \frac{p}{|I|} O(|I| d^\ell/p) = O(d^\ell)$. ◀

Proof of Lemma 23. Let $\alpha(R) = \langle A_1, \ldots, A_r \rangle$. Consider an instance I, a tuple $t \in R^I$, and a terminating path $t \to t_1 \to \cdots \to t_i$, where $i \leq k$. If $\alpha(S_i) = \langle B_1, \ldots, B_r \rangle$ is the hash signature of S_i, then t is assigned to node $m_h = h(t_i[B_1], \ldots, t_i[B_r])$. But then, since $(\lambda(e_1) \cup \cdots \cup \lambda(e_j))^{\oplus} \models R \parallel S_i$ holds, it follows that for every position ℓ, $A_\ell = B_\ell \in (\lambda(e_1) \cup \cdots \cup \lambda(e_j))^{\oplus}$. This implies that $\forall \ell, t_i[B_\ell] = t[A_\ell]$, which means that $m_h = h(\alpha(t))$. ◀

Proof of Theorem 2. For $3 \Rightarrow 2$, direct application of Lemma 23. $2 \Rightarrow 1$ is trivial. For $1 \Rightarrow 3$, let S_μ be the first relation in the root path such that $\rho_\mu \models R \parallel S_\mu$ does not hold. Let $\alpha(R) = \langle A_1, \ldots, A_r \rangle$, $\alpha(S_\mu) = \langle B_1, \ldots, B_{r'} \rangle$. We construct a tuple t_j in S_j for $j = 0, \ldots, \mu$ as follows. First, we assign to each equivalence class $\mathcal{C}$ in the equivalence relation $\rho_\mu = (\lambda(e_1) \cup \cdots \cup \lambda(e_\mu))^{\oplus}$ a distinct constant $a_{\mathcal{C}} \in \mathbf{dom}$. Then, we set $t_j[B] = a_{[B]_{\rho_\mu}}$, where $[B]_{\rho_\mu}$ is the equivalence class where B belongs.

We will now construct two instances I, I' such that for some h, $\mathbf{P}^h_{\mathbb{G}}(t_0, I) \neq \mathbf{P}^h_{\mathbb{G}}(t_0, I')$: (1) Instance I contains only the tuple t_0. In this instance t_0 will be assigned to the node $m_h = h(t_0[A_1], \ldots, t_0[A_r])$, and (2) Instance I' contains the tuples $\{t_0, t_1, \ldots, t_\mu\}$ as constructed above. Observe first that $t_{j+1} \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(t_j, I')$ for every $j = 0, \ldots, \mu - 1$. Indeed, if $(C, D) \in \lambda(e_{j+1})$, then by construction C, D belong in the same equivalence class of ρ_μ and hence we have that $t_j[C] = t_{j+1}[D]$. Thus, in instance I', t_0 will be assigned to the node $m'_h = h(t_\mu[B_1], \ldots, t_\mu[B_{r'}])$.

The key observation is that, since $\rho_\mu \models R \parallel S_\mu$ does not hold, either $r \neq r'$, or there exists $(A_i, B_i) \notin \rho_\mu$ such that A_i is not in the same equivalence class as B_i (in which case $t_0[A_i] \neq t_\mu[B_i]$). In either case, we can choose an h such that $m_h \neq m'_h$. ◀

Proof of Theorem 3. The next two lemmas prove Theorem 3. ◀

▶ **Lemma 32.** *Let $\mathbb{G} = (V, E, \alpha, \lambda)$ be a co-hash graph, relation $R \in V$, and $\mathbf{A} \subseteq \mathsf{att}(R)$. If Algorithm 1 returns true, then R is $\mathbf{A}$-clustered w.r.t. $\mathcal{P}_{\mathbb{G}}$.*

Proof. Assume that $i = \mu$ when the algorithm exits the first loop. Consider some instance I, and let T be a set of tuples in R^I such that all tuples in T agree on the attributes in $\mathbf{A}$. We will show that all tuples in T are assigned to the same unique machine. We define $J^i(T, I)$ inductively as follows, for $i = 0, \ldots, k$: $J^0(T, I) = T$, and for $i > 0$, $J^i(T, I) = \bigcup_{s \in J^{i-1}(T,I)} \mathcal{J}^{\mathbb{G}}_{\uparrow}(s, I)$. It is easy to see that by construction $J^i(T, I) \subseteq S^I_i$. First, we show the following statement using induction: *(S1) for any $i = 0, \ldots, \mu$, and any tuples $s, s' \in J^i(T, I)$ we have $s[\mathcal{V}_i] = s'[\mathcal{V}_i]$.*

For the base case, where $i = 0$, by the definition of T we have $s[\mathbf{A}] = s'[\mathbf{A}]$, and hence $s[\mathbf{A}^+] = s'[\mathbf{A}^+]$. Since $\mathcal{V}_0 = \mathbf{A}^+$, we also have $s[\mathcal{V}_0] = s'[\mathcal{V}_0]$. For the inductive step, let $s, s' \in J^i(T, I)$. By the construction of $J^i(T, I)$, there exist $r, r' \in J^{i-1}(T, I)$ such that $s \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(r, I)$ and $s' \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(r', I)$. Define $\mathcal{V}'_i = \{B \in \mathsf{att}(S_i) \mid \exists A \in \mathcal{V}_{i-1} : (A, B) \in \lambda(e_i)\}$. For any $B \in \mathcal{V}'_i$, there exists some $A \in \mathcal{V}_{i-1}$ such that $(A, B) \in \lambda(e_i)$. This implies that $s[B] = r[A]$ and $s'[B] = r'[A]$. Also, by the inductive hypothesis, it holds that $r[A] = r'[A]$. Hence, $s[B] = s'[B]$. Since $\mathcal{V}_i = (\mathcal{V}'_i)^+$, we also have $s[\mathcal{V}_i] = s'[\mathcal{V}_i]$. Second, we show: *(S2) for any $i = 0, \ldots, \mu$, and tuples $r, r' \in J^i(T, I)$, we have $\mathcal{J}^{\mathbb{G}}_{\uparrow}(r, I) = \mathcal{J}^{\mathbb{G}}_{\uparrow}(r', I)$.* Indeed, let $s \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(r, I)$. By the edge condition (EC), for every $(A, B) \in \lambda(e_{i+1})$, we have $A \in \mathcal{V}_i$. By (S1), for every $A \in \mathcal{V}_i$ we have $r[A] = r'[A]$. Thus, $s[B] = r'[A]$, which implies that s joins with r' and hence $s' \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(r, I)$ as well.

We now distinguish two cases. $\exists i \in \{1, \ldots, \mu\} : J^i(T, I) = \emptyset$. By (S2), all tuples in T are colocated with the tuples in $J^{i-1}(T, I)$. We will show that all the tuples in $J^{i-1}(T, I)$ are assigned to the same node. If $\alpha(S_{i-1}) = \langle B_1, \ldots, B_r \rangle$ is the hash signature of S_{i-1}, then any tuple $t \in J^{i-1}(T, I)$ is assigned to the node $m_h = h(t[B_1], \ldots, t[B_r])$. Some other tuple $t' \in J^{i-1}(T, I)$ is assigned to $m'_h = h(t'[B_1], \ldots, t'[B_r])$. By the vertex condition (VC), every B_j is in $\mathcal{V}_{i-1}$, and hence by (S1), $t[B_j] = t'[B_j]$, which implies that $m_h = m'_h$. Hence, all tuples in T are sent to the same unique node.

Otherwise, suppose $t \in T$ is colocated with some tuple $s \in J^\mu(T, I)$. Let $\alpha(S_\mu) = \langle A_1, \ldots, A_r \rangle$. Consider any sequence of tuples $t_1 \in S^I_{\mu+1}, t_2 \in S^I_{\mu+2}, \ldots, t_i \in S^I_{\mu+i}$, such that $t_{j+1} \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(t_j, I)$ for every $j \in \{0, \ldots, i-1\}$ and $\mathcal{J}^{\mathbb{G}}_{\uparrow}(t_i, I) = \emptyset$. If $\alpha(S_{\mu+i}) = \langle B_1, \ldots, B_r \rangle$ is the hash signature of $S_{\mu+i}$, then s (and hence t) is assigned to node $m_h = h(t_i[B_1], \ldots, t_i[B_r])$. Since by condition (HC) it holds that $(\lambda(e_1) \cup \cdots \cup \lambda(e_i))^\oplus \models S_\mu \parallel S_i$, it follows that for every position ℓ, $(A_\ell, B_\ell) \in (\lambda(e_1) \cup \cdots \cup \lambda(e_i))^\oplus$. This implies that $\forall \ell, t_i[B_\ell] = s[A_\ell]$. Hence, tuple t is sent to node $m_h = h(s[A_1], \ldots, s[A_r])$. If $t' \in T$ is some other tuple colocated with some $s' \in J^\mu(T, I)$, then similarly we can argue that it is sent to node $m'_h = h(s'[A_1], \ldots, s'[A_r])$. But since for every attribute A_j we have $A_j \in \mathcal{V}_\mu$, it holds that $s[A_j] = s'[A_j]$ and hence $m_h = m'_h$. ◀

▶ **Lemma 33.** *Let $\mathbb{G} = (V, E, \alpha, \lambda)$ be a co-hash graph, relation $R \in V$, and $\mathbf{A} \subseteq \mathsf{att}(R)$. If Algorithm 1 returns false, then there h such that R is not $\mathbf{A}$-clustered w.r.t. $\mathbf{P}^h_{\mathbb{G}}$.*

Proof. We distinguish two cases, depending on whether only (VC) fails, or both (EC) and (HC) fail together. Suppose that the algorithm returns false because (VC) fails for some relation S_μ, $\mu \geq 0$. We then construct an instance I with labelled nulls as follows. First, we assign to each equivalence class C in the equivalence relation $(\lambda(e_1) \cup \ldots \cup \lambda(e_{\mu-1}))^{\oplus}$ two distinct nulls $\perp_C, \perp'_C$. Every relation S_j for $j = 0, \ldots, \mu$ contains two tuples t_j, t'_j such that for any attribute $A \in C$, $t_j[A] = t'_j[A] = \perp_C$ whenever $A \in \mathcal{V}_j$, otherwise $t_j[A] = \perp_C$ and $t'_j[A] = \perp'_C$. All the other relations are empty. It is easy to see that I satisfies the functional dependencies, and also t_0, t'_0 agree on $\mathbf{A}$. Moreover, for every $j < \mu$, we have $t_{j+1} \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(t_j, I)$ and $t'_{j+1} \in \mathcal{J}^{\mathbb{G}}_{\uparrow}(t'_j, I)$. Let $\alpha(S_\mu) = \langle B_1, \ldots, B_r \rangle$. Then, tuple t_0 is sent to $h(t_\mu[B_1], \ldots, t_\mu[B_r])$ and tuple t'_0 to $h(t'_\mu[B_1], \ldots, t'_\mu[B_r])$. Since (VC) failed at S_μ, there exists some attribute $B_k \notin \mathcal{V}_\mu$, in which case $t_\mu[B_k] \neq t'_\mu[B_k]$. Hence, we can always choose a hash function h such that $\mathbf{P}^h_{\mathbb{G}}(t_0, I) \neq \mathbf{P}^h_{\mathbb{G}}(t'_0, I)$.

Suppose that the algorithm returns false because (EC) and (HC) fail together. Let $\mu \geq 0$ be the first index with join condition $\lambda(e_{\mu+1})$ for which (EC) fails. Moreover, let S_l, $l > \mu$, be the first relation where condition (HC) fails. We construct an instance I with labelled nulls as follows. First, we assign to each equivalence class C in the equivalence relation $(\lambda(e_1) \cup \ldots \cup \lambda(e_{\ell-1}))^{\oplus}$ two distinct nulls $\perp_C, \perp'_C$. We construct two types of tuples in I: (1) tuples t_j in S_j for $j = 0, \ldots, \ell$. For an attribute $B \in C$, we set $t_j[B] = \perp_C$, and (2) tuples t'_j in S_j for $j = 0, \ldots, \mu$. For an attribute $B \in C$, we set $t'_j[B] = \perp_C$ if $B \in \mathcal{V}_j$, otherwise $t'_j[B] = \perp'_C$.

It is easy to see that t_0, t'_0 agree on the attributes in $\mathbf{A}$. Moreover, t_0 will be colocated with t_ℓ, and t'_0 will be colocated with t'_μ. Now, since condition (EC) failed, there exists $(A, B) \in \lambda(e_{\mu+1})$ such that $A \notin \mathcal{V}_\mu$. Suppose that A belongs in the equivalence class C. By our construction, this implies that $t'_\mu[A] = \perp'_C$, but $t_{\mu+1}[B] = \perp_C$. Hence, $\mathcal{J}^{\mathbb{G}}_{\uparrow}(t'_\mu, I) = \emptyset$. Let $\alpha(S_\mu) = \langle A_1, \ldots, A_r \rangle$ and $\alpha(S_\ell) = \langle B_1, \ldots, B'_r \rangle$. Then, t_0 is sent to $m_h = h(t_\ell[B_1], \ldots, t_\ell[B'_r])$, and t'_0 to $m_h = h(t'_\mu[A_1], \ldots, t'_\mu[A_r])$. Moreover, since (VC) holds for S_μ, every $A_j \in \mathcal{V}_\mu$, and hence $m_h = h(t_\mu[A_1], \ldots, t_\mu[A_r])$. Finally, since S_μ, S_ℓ are not hash-compatible w.r.t. $(\lambda(e_1) \cup \ldots \cup \lambda(e_{\ell-1}))^{\oplus}$, either $r \neq r'$, or there exists a position j such that (A_j, B_j) is not in the equivalence relation, which would imply that $t_\ell[B_j] \neq t_\mu[A_j]$. In both cases, we can pick a hash function h such that t_0, t'_0 end up in different nodes. ◀

Proof of Theorem 4. To show membership in Π^P_2, we will show that the complement is in Σ^P_2. To this end, we will give a polynomial time algorithm with the following property: there exists a valuation v such that for every valuation v' the algorithm returns yes for (v, v') if and only if q is not p-correct on I. First, the algorithm checks whether v, v' satisfy q. If v does not satisfy q, it returns no. If v' does not satisfy q, it returns yes. Then, it checks that v, v' agree on the head of q, and if not, it returns yes. Finally, it checks whether the tuples $\{v'(\mathbf{x}_1), \ldots, v'(\mathbf{x}_\ell))$ are collocated in instance I. If not, it returns yes, otherwise it terminates with no. It is easy to see that all the checks can be done in polynomial time following from Lemma 17.

To show Π^P_2-hardness, we construct a reduction from the problem PCI($\mathcal{P}_{\text{fin}}$), defined in [3]. In this problem, we are given a CQ q, an instance I, and a distribution policy $\mathbf{P}$ which is explicitly enumerated as part of the input (i.e., for each tuple in I we know its destination node). Then, it asks whether q is p-correct in I under $\mathbf{P}$.

We construct a co-hash graph $\mathbb{G} = (V, E, \alpha, \lambda)$ as follows. For every relation $R(A_1, \ldots, A_k)$ in q, we introduce two nodes in V: one is R, and the other is a fresh relation $R'(C, A'_1, \ldots, A'_k)$. E contains edges of the form (R, R'), where $\lambda((R, R')) = \{A_1 = A'_1, \ldots, A_k = A'_k\}$. Finally,

we set $\alpha(R) = \langle\rangle$ and $\alpha(R') = \langle C\rangle$. For this schema, we create an instance I', where for every tuple $t = R(a_1, \ldots, a_k) \in I$ we add the tuples $R(a_1, \ldots, a_k)$ and $\{R'(\kappa, a_1, \ldots, a_k) \mid \kappa \in \mathbf{P}(t)\}$ to I'. Intuitively, each relation R' encodes the destinations of the tuples in R according to $\mathbf{P}$.

The reduction is in polynomial time. We now claim that q is p-correct in I under $\mathbf{P}$ if and only if q is p-correct on I' under $\mathcal{P}_{\mathbb{G}}$. Indeed, notice that for some hash function h, tuple $t \in I$ gets assigned to $\{h(\kappa) \mid \kappa \in \mathbf{P}(t)\}$. Hence, if for a set of tuples $T \subseteq I$ we have $\kappa \in \bigcap_{t\in T} \mathbf{P}(t)$, then $h(\kappa) \in \bigcap_{t\in T} \mathbf{P}^h_{\mathbb{G}}(t, I')$ for any h, which implies $I' \triangleright_{\mathbb{G}} T$. If $\bigcap_{t\in T} \mathbf{P}(t) = \emptyset$, then the hash function $h(\kappa) = \kappa$ implies that $I' \not\triangleright_{\mathbb{G}} T$. ◀

Proof of Theorem 5. We first show membership in coNP. Let $q(\mathbf{y}) = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell)$ be a full CQ, and an instance I. We will guess a valuation v over the variables of q, and then check that (i) for every i, $v(\mathbf{x}_i) \in I$, and (ii) $I \triangleright_{\mathbb{G}} \{v(\mathbf{x}_1), \ldots, v(\mathbf{x}_\ell)\}$. Indeed, any such valuation will be a witness that p-correctness is violated. It remains to show that (i) and (ii) can be done in polynomial time. Indeed, (i) is straightforward. For (ii), we can apply directly Lemma 17.

We show coNP-hardness by reduction from the problem of CQ evaluation. Consider a boolean CQ $q() = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell)$, an instance I, and the problem that asks whether $q(I)$ is empty or not, which is known to be NP-hard. We construct a full query q' from q as follows: $q'(\ldots) = S_1(z_1), R'_1(z_1, \mathbf{x}_1), \ldots, S_\ell(z_\ell), R'_\ell(z_\ell, \mathbf{x}_\ell)$, where $\{z_1, \ldots, z_\ell\}$ are distinct fresh variables. To construct an instance I', for every tuple $R_i(\mathbf{a}_i) \in I$, we introduce $R'_i(i, \mathbf{a}_i) \in I'$. Note that this means that a tuple in I can create many copies of it in I' (as many as the number of times its relation occurs in q). Moreover, for every $i = 1, \ldots, \ell$, we add the tuple $S_i(i)$. It is easy to verify that $q(I)$ is true if and only if $q'(I') \neq \emptyset$. Finally, we construct a co-hash graph $\mathbb{G} = (V, E, \alpha, \lambda)$ as follows: V contains all relations in the body of q', $E = \emptyset$, and α uses only the first attribute of each relation for hashing. Note that $\mathcal{P}_{\mathbb{G}}$ is an oblivious distribution scheme!

We now show the following: q' is p-correct on I' under $\mathcal{P}_{\mathbb{G}}$ if and only if $q(I)$ is false. For the one direction, suppose $q(I)$ is false. Then, $q'(I') = \emptyset$. Hence, no matter how $\mathcal{P}_{\mathbb{G}}$ distributes I', we trivially have p-correctness. For the other direction, suppose $q(I)$ is true. Then, there exists a valuation v over the variables of q that makes it true. This can be extended to a valuation v', where $v'(z_i) = i$, that makes q' true as well. Let $t_1, t_2, \ldots, t_\ell \in I'$ be the tuples that correspond to v' for $R'_1, \ldots, R'_\ell$. These are all different (since their first attribute must be different). Moreover, t_i is assigned to $h(i)$. This means we can choose an h such that every t_i goes to a different node, which shows that q' is not p-correct. ◀

Proof of Theorem 6. We show a reduction from the problem of query containment: given as input two CQs q_1, q_2, is q_1 contained in q_2? Without any loss of generality, we assume that both queries are boolean and use one binary relation $R(A, B)$.

We construct a boolean CQ q as input to CPC as follows. The schema Σ contains two relations, $R'(C, A, B)$ and $T(D)$. For every atom $R(x, y)$ in q_1, q contains the atom $R'(w^{(1)}, x^{(1)}, y^{(1)})$, and for every atom $R(x, y)$ in q_2 it contains the atom $R'(w^{(2)}, x^{(2)}, y^{(2)})$. Moreover, we add an atom $T(w^{(1)})$. Let q'_1 (q'_2 resp.) be the subquery of q that contains atoms with variable $w^{(1)}$ ($w^{(2)}$ resp.). The input co-hash graph is simple: the hash signature for R' is $\langle C\rangle$ and for T is $\langle D\rangle$, and it contains no edges.

We now claim that q is p-correct under $\mathcal{P}_{\mathbb{G}}$ if and only if $q_1 \subseteq q_2$. For the one direction, assume that $q_1 \subseteq q_2$. Then, it is easy to see that q is equivalent to q'_1. Consider any valuation v that satisfies q'_1: then all the relevant facts will end up in the same hash destination $h(v(w^{(1)}))$, making the query p-correct. For the opposite direction, assume that $q_1 \not\subseteq q_2$. Let I be the canonical database of q, so $q(I)$ is true. By our construction, we have $D[q] = D[q'_1] \cup D[q'_2]$

and $D[q_1'] \cap D[q_2'] = \emptyset$. The facts of $D[q_1']$ will end up on node $\kappa_1 = h(w^{(1)})$, while the facts of $D[q_2']$ on node $\kappa_2 = h(w^{(2)})$. Additionally, we can always choose the hash function h such that $\kappa_1 \neq \kappa_2$. Since $D[q_2']$ does not have any T-facts, we have that $q(D[q_2'])$ is false. Since $q_1 \not\subseteq q_2$, we also have that $q(D[q_1'])$ is false. Thus, I falsifies the p-correctness property. ◀

Proof of Theorem 7. To analyze the runtime, we first note that the canonical instance has at most as many tuples as the number of atoms ℓ in the body of the query. Moreover, for every such tuple, the while loop will terminate after at most k iterations, where k is the length of the longest root path in $\mathbb{G}$ ($k \leq |E|$). This means that the size of any instance I_i generated by the algorithm is always polynomially bounded (in fact, it will be at most $|E| + \ell$). The check $I_i \triangleright_{\mathbb{G}} D_i$ can be done in polynomial time using Lemma 17, while the extension step also runs in polynomial time.

For correctness, we claim that Algorithm 2 outputs true if and only if the query q is p-correct under $\mathcal{P}_{\mathbb{G}}$. The one direction is straightforward. Indeed, if there exists an instance I_i such that $D_i \subseteq I_i$ and the tuples in D_i are not collocated, then I_i is a witness that p-correctness is violated (since any D_i forms an answer for $q(I_i)$).

The other direction is more involved. Consider an instance I and any valuation v such that $t_i = v(\mathbf{x}_i) \in I$ for every $i = 1, \ldots, \ell$. Let $T = \{t_1, t_2, \ldots, t_\ell\}$. To prove that the query q is p-correct, it suffices to show that $I \triangleright_{\mathbb{G}} T$. Let $T_\square \subseteq T$ be the set of tuples such that none of their terminating paths go through a tuple in T. It is easy to check that $T_\square \neq \emptyset$. It now suffices to show that $I \triangleright_{\mathbb{G}} T_\square$.

If there exists a tuple $s \in I$ such that every $t \in T_\square$ has a terminating path that goes through s, then the claim $I \triangleright_{\mathbb{G}} T_\square$ follows directly. For the remainder of the proof, we assume that this is not the case. Since $D[q]$ is the canonical instance for q, there exists a strong[1] homomorphism $\xi : D[q] \to T$. Since the homomorphism is strong, for every $t \in T$ there exists a tuple $\tilde{t} \in D[q]$ such that $\xi(\tilde{t}) = t$. Note that all tuples in the set $\{\tilde{t} \mid t \in T_\square\}$ must have an empty upwards join set in $D[q]$. Since $D[q] \triangleright_{\mathbb{G}} D[q]$, it must be that all $\alpha(\tilde{t})$ are equal to the same vector β (consisting of labelled nulls and constants) for every $t \in T_\square$. But then, $\alpha(t) = \alpha(\xi(\tilde{t})) = \xi(\beta)$. Consider any $t \in T_\square$ and a terminating path $(t =)s_0 \to s_1 \to \cdots \to s_k$ in I. By our construction, none of the tuples $s_1, \ldots, s_k$ are in T. We will show that $\alpha(s_k) = \xi(\beta)$; this implies that all tuples in $T_\square$ are collocated in $h(\xi(\beta))$ for any hash function h, hence proving the claim. We start with two observations: (1) The strong homomorphism ξ can be extended to a strong homomorphism from the instance I_k generated by the algorithm for the while loop of $\tilde{t}$ to the instance $J = T \cup \{s_1, \ldots, s_k\}$, and (2) Let $\phi = \phi_1 \circ \phi_2 \cdots \circ \phi_k$. Then, for any hash destination γ (over labelled nulls and constants) $\xi(\phi(\gamma)) = \xi(\gamma)$.

Note that some tuple $u \in T_\square$ must have an empty upwards join set in J; otherwise, every tuple in $T_\square$ has a terminating path that goes through s_k, a contradiction. Hence, the tuple $\tilde{u} \in I_k$ with $\xi(\tilde{u}) = u$ has an empty upwards join set in I_k, which implies that $\alpha(\tilde{u}) = \phi(\beta)$ is its only hash destination. We can also see that $\tilde{t}$ has a unique hash destination in I_k, $\alpha(\tilde{s}_k)$. Since $\tilde{u}, \tilde{t}$ must be collocated to guarantee p-correctness for the instance I_k, it must be that $\alpha(\tilde{s}_k) = \alpha(\tilde{u}) = \phi(\beta)$. Thus, $\alpha(s_k) = \xi(\phi(\beta)) = \xi(\beta)$, where the last equality is implied by the second observation. ◀

Proof of Theorem 8. To show membership in Π_2^P, we will show that the complement is in Σ_2^P. To this end, we will give a polynomial time algorithm with the following property: there exists a valuation v such that for every valuation v' the algorithm returns yes for (v, v') if

[1] A homomorphism $h : I \to I'$ is strong if for every tuple $t \in I'$, there exists a tuple $s \in I$ such that $t = h(s)$.

and only if q is not p-correct on I. First, the algorithm checks whether v, v' satisfy q. If v does not satisfy q, it returns no. If v' does not satisfy q, it returns yes. Then, it checks that v, v' agree on the head of q, and if not, it returns yes. Finally, it checks whether for some hash family h, $|\bigcap_i \mathbf{P}^h_{\mathbb{G}}(v'(\mathbf{x}_i), I)| \neq 1$. If not, it returns yes, otherwise it terminates with no. It is easy to see that all the checks can be done in polynomial time from Lemma 17. Indeed, we can modify the construction in Lemma 17 so that is computes the intersection of the hash destinations as well (instead of only checking for emptiness).

To show Π^P_2-hardness, we first notice that the hardness proof for the problem PCI($\mathcal{P}_{\text{fin}}$) in [3] can be modified to show Π^P_2-hardness for the pd-correct variant of the problem. Indeed, all p-correct instances for the proof in [3] are also pd-correct. Formally, we define PDCI($\mathcal{P}_{\text{fin}}$) as follows: we are given a CQ q, an instance I, and a distribution policy $\mathbf{P}$ which is explicitly enumerated as part of the input (i.e., for each tuple in I we know its destination node). Then, it asks whether q is pd-correct in I under $\mathbf{P}$. The reduction is exactly the same as the one in Theorem 4. We construct a co-hash graph $\mathbb{G} = (V, E, \alpha, \lambda)$ as follows. For every relation $R(A_1, \ldots, A_k)$ in q, we introduce two nodes in V: one is R, and the other is a fresh relation $R'(C, A'_1, \ldots, A'_k)$. E contains edges of the form (R, R'), where $\lambda((R, R')) = \{A_1 = A'_1, \ldots, A_k = A'_k\}$. Finally, we set $\alpha(R) = \langle\rangle$ and $\alpha(R') = \langle C\rangle$.

For this schema, we create an instance I', where for every tuple $t = R(a_1, \ldots, a_k) \in I$ we add the tuples $R(a_1, \ldots, a_k)$ and $\{R'(\kappa, a_1, \ldots, a_k) \mid \kappa \in \mathbf{P}(t)\}$ to I'. Intuitively, each relation R' encodes the destinations of the tuples in R according to $\mathbf{P}$. We now claim that q is pd-correct in I under $\mathbf{P}$ if and only if q is pd-correct on I' under $\mathcal{P}_{\mathbb{G}}$. Indeed, notice that for some hash function h, tuple $t \in I$ gets assigned to $\{h(\kappa) \mid \kappa \in \mathbf{P}(t)\}$. Hence, if for a set of tuples $T \subseteq I$ we have $\kappa \in \bigcap_{t \in T} \mathbf{P}(t)$, then $h(\kappa) \in \bigcap_{t \in T} \mathbf{P}^h_{\mathbb{G}}(t, I')$ for any h. The result follows by picking $h(\kappa) = \kappa$.

The coNP-hardness proof follows the same structure as the one for p-correctness (Theorem 5). Indeed, in the reduction p-correctness holds only if the query result is empty, in which case pd-correctness also trivially holds. To show membership in coNP, consider a full query $q(\mathbf{y}) = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell)$ and an instance I. We guess a valuation v over the variables of q, and then check that (i) for every i, $v(\mathbf{x}_i) \in I$, (ii) for some hash family h, $|\bigcap_i \mathbf{P}^h_{\mathbb{G}}(v(\mathbf{x}_i), I)| \neq 1$. Indeed, any such valuation will be a witness that pd-correctness is violated.

It remains to show that (i) and (ii) can be done in polynomial time. Indeed, (i) is straightforward. For (ii), we can modify the construction in Lemma 17 so that is computes the intersection of the hash destinations as well (instead of only checking for emptiness). ◀

Proof of Theorem 9. It is easy to observe that if any relation in a query is non-redundant (which we can check using Algorithm 1 by setting $\mathbf{A} = \mathsf{att}(R)$), then p-correctness implies pd-correctness. However, this condition is not necessary. To address this issue, we consider a generalization of non-redundancy from relations to atoms. Given a co-hash graph $\mathbb{G}$ and a CQ q, we say that an atom $R_i(\mathbf{x}_i)$ in q is *dominant* if the root path of R_i contains no other relation that appears in the body of q. To compute whether a dominant atom $R_i(\mathbf{x}_i)$ is non-redundant, we simply modify Algorithm 1 such that the closure in Line 16 includes $A = A'$ whenever two attributes A, A' of R_i have the same variable in $R_i(\mathbf{x}_i)$. Now, let $\mathbb{G}$ be a co-hash graph, and q a full CQ such that q is p-correct under $\mathcal{P}_{\mathbb{G}}$. Then, q is pd-correct under $\mathcal{P}_{\mathbb{G}}$ iff there exists a dominant atom in q that is non-redundant.

One direction of the proof is straightforward. If some dominant atom $R_i(\mathbf{x}_i)$ is non-redundant, then the tuples for any valuation v must meet on the unique location where $v(\mathbf{x}_i)$ is assigned to. For the other direction, suppose that $R_i(\mathbf{x}_i)$ is a dominant and redundant atom. Let $D[q]$ be the canonical instance for q with labelled nulls (and constants). Let

$t \in D[q]$ be the tuple corresponding to the atom $R_i(\mathbf{x}_i)$. Following the proof in Lemma 33, we can construct an instance $I = P_1 \cup P_2$ such that $t \in I$, and P_1, P_2 are two terminating paths for t in I with hash destinations h_1 and h_2 respectively, where $h_1 \neq h_2$. Consider now two instances: $I_1 = D[q] \cup P_1$ and $I_2 = D[q] \cup P_2$. Since there is no other relation of q in the root path of R_i, the tuple t must have h_1, h_2 as its only hash destinations in I_1, I_2 respectively. Moreover, since q is p-correct, it must be that all tuples in $D[q]$ have the same hash destination h_1 in I_1, and h_2 in I_2.

Finally, consider the instance $I' = D[q] \cup P_1 \cup P_2$. We can now argue that all tuples of $D[q]$ will end up in both h_1, h_2, thus proving that the query is not pd-correct. Indeed, suppose for the sake of contradiction that they end up in h_1 but not h_2. Then, there exists a tuple $s \in D[q]$ that is assigned to h_2 in I_2, but not in I'. For this to have happened, it must be the case that some tuple $s' \in D[q]$ has an empty upwards join in I_2, but not in I', where it joins with some tuple from P_1. But then, the hash signature of s' is equal to h_2. Since P_1, P_2 differ only in their non-join values, this implies that s' would be equal to h_1 as well, so $h_1 = h_2$, a contradiction. Since we can check non-redundancy in polynomial time, the above algorithm implies a polynomial time algorithm for pd-correctness. ◀