25th International Conference on Database Theory

ICDT 2022, March 29–April 1, 2022, Edinburgh, UK
(Virtual Conference)

Edited by

Dan Olteanu
Nils Vortmeier

Editors

Dan Olteanu
University of Zurich, Switzerland
dan.olteanu@uzh.ch

Nils Vortmeier
University of Zurich, Switzerland
nils.vortmeier@uzh.ch

ACM Classification 2012
Information systems → Data management systems; Information systems → Database design and models; Information systems → Database query processing; Information systems → Database transaction processing; Information systems → Join algorithms; Information systems → Query languages; Information systems → Query optimization; Information systems → Query planning; Information systems → Relational database model; Information systems → Parallel and distributed DBMSs; Information systems → Semantic web description languages; Information systems → Stream management; Theory of computation → Approximation algorithms analysis; Theory of computation → Complexity theory and logic; Theory of computation → Constraint and logic programming; Theory of computation → Incomplete, inconsistent, and uncertain databases; Theory of computation → Database theory; Theory of computation → Data structures and algorithms for data management; Theory of computation → Description logics; Theory of computation → Finite Model Theory; Theory of computation → Logic and databases; Theory of computation → Problems, reductions and completeness; Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Random number generation

ISBN 978-3-95977-223-5

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/978-3-95977-223-5.

Publication date
March, 2022

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.2022.0
ISBN 978-3-95977-223-5 **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

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The 25. International Conference on Database Theory (ICDT 2022) was held from March 29 to April 1, 2022, as an online event.

The Program Committee has selected 16 research papers out of 41 submissions for publication at the conference. It has further decided to give the Best Paper Award to *On the Hardness of Category Tree Construction* by Shay Gershtein, Uri Avron, Ido Guy, Tova Milo, and Slava Novgorodov, and the Best Newcomer Paper Award to *Linear programs with conjunctive queries* by Florent Capelli, Nicolas Crosetti, Joachim Niehren, and Jan Ramon. We congratulate the winners!

Apart from the 16 regular papers, these proceedings include abstracts for the invited (shared) EDBT/ICDT keynote by Marcelo Arenas (Pontificia Universidad Católica de Chile) and for the ICDT invited tutorial by Nofar Carmeli (École Normale Supérieure Paris), as well as the invited paper associated with the (shared) EDBT/ICDT keynote by Hung Ngo (RelationalAI Inc.).

A committee formed by Antoine Amarilli, Alin Deutsch, and Emanuel Sallinger has decided to give the Test-of-Time Award for ICDT 2022 to the ICDT 2012 paper *Factorised Representations of Query Results: Size Bounds and Readability* by Dan Olteanu and Jakub Závodný.

We would like to thank all people who contributed to the success of ICDT 2022, including the authors of all submitted papers, keynote and invited tutorial 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 2022. 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, and the local organizers of the EDBT/ICDT 2022 conference, led by General Chairs Paolo Guagliardo, Milos Nikolic, and Andreas Pieris, 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.

Dan Olteanu and Nils Vortmeier
March 2022

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

General Chairs

Paolo Guagliardo (University of Edinburgh)

Milos Nikolic (University of Edinburgh)

Andreas Pieris (University of Edinburgh)

Program Chair

Dan Olteanu (University of Zurich)

Program Committee

Mahmoud Abo Khamis (RelationalAI Inc.)

Suman Bera (UC Santa Cruz)

Nofar Carmeli (École Normale Supérieure Paris)

Ismail Ilkan Ceylan (University of Oxford)

Daniel Deutch (Tel Aviv University)

Cristina Feier (Bremen University)

Xiao Hu (Duke University)

Ahmet Kara (University of Zurich)

Egor Kostylev (University of Oslo)

Andrew McGregor (University of Massachusetts, Amherst)

Mikaël Monet (Inria Lille)

Hung Ngo (RelationalAI Inc.)

Liat Peterfreund (École Normale Supérieure Paris)

Reinhard Pichler (Vienna University of Technology)

Cristian Riveros (Pontificia Universidad Católica de Chile)

Thomas Schwentick (TU Dortmund University)

Dan Suciu (University of Washington)

Yufei Tao (The Chinese University of Hong Kong)

Szymon Toruńczyk (University of Warsaw)

Proceedings Chair

Nils Vortmeier (University of Zurich)

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Sabyasachi Basu

Michael Benedikt

Arnaud Durand

Antonia Kormpa

Victor Marsault

Stefan Mengel

Frank Neven

Matthias Niewerth

Milos Nikolic

Florin Rusu

Luc Segoufin

Christopher Spinrath

Sławek Staworko

Nils Vortmeier

Domagoj Vrgoč

Chenghong Wang

Yisu Wang

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Contributing Authors

Raghavendra Addanki

Marcelo Arenas

Uri Avron

David Basin

Bart Bogaerts

Florent Capelli

Nofar Carmeli

Nicolas Crosetti

Austen Z. Fan

Cristina Feier

Dominik D. Freydenberger

Gaetano Geck

Shay Gershtein

Benoît Groz

Ido Guy

Maxime Jakubowski

Jens Keppeler

Bas Ketsman

Sarah Kleest-Meißner

Christoph Koch

Paraschos Koutris

Markus Krötzsch

Srđan Krstić

Aurélien Lemay

Maximilian Marx

Andrew McGregor

Alexandra Meliou

Jingfan Meng

Tova Milo

Zafeiria Moumoulidou

Martín Muñoz

Frank Neven

Hung Q. Ngo

Joachim Niehren

Slava Novgorodov

Mitsunori Ogihara

Jan Ramon

Martin Raszyk

Cristian Riveros

Rebecca Sattler

Markus L. Schmid

Nicole Schweikardt

Thomas Schwentick

Christopher Spinrath

Sławek Staworko

Yufei Tao

Sam M. Thompson

Dmitriy Traytel

Jan Van den Bussche

Brecht Vandevoort

Huayi Wang

Matthias Weidlich

Piotr Wieczorek

Jun Xu

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The ICDT 2022 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 earlier ICDT conferences that have best met the "test of time". In 2022, the award recognizes a paper selected from the proceedings of the ICDT 2012 conference that has had the highest impact in terms of research, methodology, conceptual contribution, or transfer to practice over the past decade. The award was presented during the EDBT/ICDT 2022 Joint Conference, March 29 – April 1st, 2022.

The 2022 ToT Committee consists of Antoine Amarilli, Alin Deutsch, and Emanuel Sallinger. After careful consideration and soliciting external assessments, the committee has chosen the following contribution for the 2022 ICDT Test-of-Time Award:

Factorised Representations of Query Results: Size Bounds and Readability
Dan Olteanu and Jakub Závodný

This paper introduced the fundamental concept of *factorized data representations*, subsequently studied as *factorized databases* in database theory. Factorized representations avoid redundancy in relations by building them up from singleton tuples using the union and product operators. The work by Olteanu and Závodný gave a formal definition of these factorized representations and showed their numerous benefits: they are more succinct than flat relations, can be used to evaluate queries more efficiently, and their tuples can be enumerated with linear-time preprocessing and constant delay. Their work further shows size bounds on factorized representations of query results depending on graph-theoretic parameters of the query, and also considers many extensions, such as factorized provenance representations, connections to readability width, and aggregation over factorized representations.

Since then, this concept of factorized databases has had profound impact on several areas of database theory, database systems research, and neighboring areas. Applications include query evaluation over graph databases, improved enumeration results, factorized computation of aggregates, and factorized machine learning. Their work has successfully bridged the gap between theory and practice: it has prompted implementations of its core ideas in subsequent works, and it has also sparked practical research across several independent areas.

Factorized databases are acknowledged as an inspiration within many lines of practical and theoretical research, including the work on aggregate queries by Abo Khamis et al. (PODS 2016 best paper award), the work on SPARQL by Abul-Basher et al. (EDBT 2021 best short paper award), and many others. This conclusively demonstrates the lasting influence of this highly cited paper and of its full version titled "Size Bounds for Factorised Representations of Query Results" which later appeared in *ACM Transactions on Database Systems*.

Antoine Amarilli, Télécom Paris
Alin Deutsch, UC San Diego
Emanuel Sallinger, TU Wien and Oxford

The ICDT Test-of-Time Award Committee for 2022

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On an Information Theoretic Approach to Cardinality Estimation

Hung Q. Ngo
RelationalAI Inc., Berkeley, CA, USA

Abstract

This article is a companion to an invited talk at ICDT'2022 with the same title.

Cardinality estimation is among the most important problems in query optimization. It is well-documented that, when query plans go haywire, in most cases one can trace the root cause to the cardinality estimator being far off. In particular, traditional cardinality estimation based on selectivity estimation may sometimes under-estimate cardinalities by orders of magnitudes, because the independence or the uniformity assumptions do not typically hold.

This talk outlines an approach to cardinality estimation that is "model-free" from a statistical stand-point. Being model-free means the approach tries to avoid making any distributional assumptions. Our approach is information-theoretic, and generalizes recent results on worst-case output size bounds of queries, allowing the estimator to take into account histogram information from the input relations. The estimator turns out to be the objective of a maximization problem subject to concave constraints, over an exponential number of variables. We then explain how the estimator can be computed in polynomial time for some fragment of these constraints. Overall, the talk introduces a new direction to address the classic problem of cardinality estimation that is designed to circumvent some of the pitfalls of selectivity-based estimation. We will also explain connections to other fundamental problems in information theory and database theory regarding information inequalities.

The talk is based on (published and unpublished) joint works with Mahmoud Abo Khamis, Sungjin Im, Hossein Keshavarz, Phokion Kolaitis, Ben Moseley, XuanLong Nguyen, Kirk Pruhs, Dan Suciu, and Alireza Samadian Zakaria

2012 ACM Subject Classification Information systems → Query optimization; Information systems → Query planning; Information systems → Join algorithms

Keywords and phrases Cardinality Estimation, Information Theory, Polymatroid Bound, Worst-case Optimal Join

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.1

Category Invited Talk

Acknowledgements I would like to thank the technical program committee and the program chair Dan Olteanu of ICDT 2022 for inviting me to give a keynote talk at the conference

1 Introduction

Cardinality estimation [25] is a crucial component of the query optimization pipeline. The problem can be formulated as shown in Figure 1. We collect statistical profiles $s(\boldsymbol{D})$ of relations in the database $\boldsymbol{D}$. (Profiles are also called "system catalogs" [51], among other names.) For a given query Q, the problem is to come up with an accurate estimate $\hat{q}$ of $|Q(\boldsymbol{D})|$ from the profile $s(\boldsymbol{D})$, as quickly as possible. The main tradeoff dimensions are the *space complexity* of $s(\boldsymbol{D})$, the *speed* of computing $\hat{q}$, and the *accuracy* of the estimate.

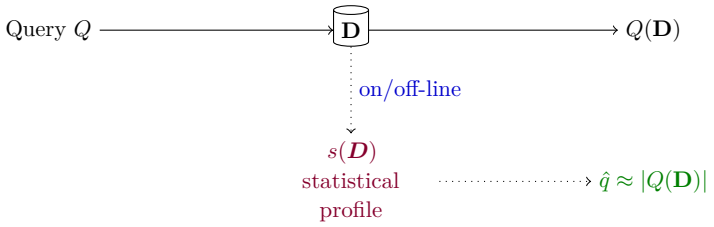


Figure 1 Cardinality Estimation.

Cardinality estimation is one of the most, if not the most, important component of the query optimization pipeline [38, 42, 40]. Cardinality estimates are the main parameters in the cost-estimators of logical query plans, physical query plan, parallel query processing, and in computing budgets for in-memory query processing [1]. Guy Lohman [42] expressed the issue succinctly:

> *The root of all evil*, the Achilles Heel of query optimization, is the estimation of the size of intermediate results, known as *cardinalities*. Everything in cost estimation depends upon how many rows will be processed, so the entire cost model is predicated upon the cardinality model. ... In my experience, the cost model may introduce errors of at most 30% for a given cardinality, but the cardinality model can quite *easily introduce errors of many orders of magnitude!*

Cardinality estimation is a very challenging problem. After all, the problem is fundamentally a lossy compression problem: different databases may have the same or similar (small) statistical profiles, how do we tell them apart? After more than half a century of theory and implementation of relational database systems, whose global market size is more than 60 billions USD in 2020[1], commercial database systems still routinely misestimate cardinalities by a factor of 1000 or more [38]. Two major reasons for the misestimation are: (1) relational RDBMSs employ estimators that make distributional assumptions about the data (such as uniformity) which may not hold in real workloads or in standard benchmarks, and (2) traditional estimators treat selection predicates independently, leading to error accumulation on large queries. Hence, estimation errors grow exponentially as the number of joins increases [30, 40]. The lack of whole-query constraints consideration also led to strange phenomena where "simply by swapping predicates or relations", the estimates can change drastically [38].

A natural question is, "how can database theory be (more) helpful in addressing these challenges?" There are a steady stream of works from the database systems community studying the cardinality estimation problem, year after year (see [43, 43, 11, 39, 26, 36, 19, 57, 25, 44] and references thereoff). Yet there are remarkably much fewer works from the database theory community on this fundamental problem. The Alice book [3] does not have any mention of cardinality estimation (theory or otherwise). The few database theory papers related to cardinality estimation were on aspects of an existing approach, and the vast majority of them were published in the 1990s [18, 52, 41, 12, 37, 13, 31, 24, 21, 23, 16]. There are very few papers, if any at all, that attempt to take a fresh look at cardinality estimation.

[1] https://www.gartner.com/en/documents/4001330/market-share-database-management-systems-worldwide-2020

This talk presents an cardinality estimation approach that is developed and implemented at RelationalAI Inc. (with collaborators). The approach is aimed to address both of the weaknesses mentioned above. The first weakness is the (strong) model-based assumptions which leading to the lack of robustness. The second weakness is the one-selection-at-a-time estimation strategy which leads to error accumulation, in part due to its inability to take into account statistical information available in the query but *outside* the scope of the selection predicate being considered.

Before describing the approach, we describe the cardinality estimation problem more formally. We shall start with the profile $s(\boldsymbol{D})$, and then describe different types of estimators $\hat{q}$, the two components shown in Figure 1.

There are several types of (statistical) profiles $s(\boldsymbol{D})$, which can be collected offline or online (after seeing the query Q). In the offline case, the simplest profiles involve integrity constraints such as functional dependencies, number of distinct values of a given attribute, and base-table cardinalities and number of disk pages. More complex profiles include synopses such as histograms, approximate histograms, frequency moments [29, 25] of degree sequences, moments and quantiles for numeric variables, trie or bigrams/trigrams for string variables, and so on [15]. In the online case, $s(\boldsymbol{D})$ can contain samples from the database, constructed based on the query Q. Sampling-based estimator is a deep topic both theoretical and practically [40, 19, 39, 41, 27]. More recently, machine learning based approaches are starting to show up in the literature [57, 36], but the jury is still out on whether they are practical enough to be in production.

This talk concentrates on the *offline* case, where $s(\boldsymbol{D})$ contains some forms of histogram information and integrity constraints. The setup is sufficiently simple for a fresh look at the problem, yet powerful enough to capture and even generalize standard settings in most RDBMSs. Simplicity also leads to practicality.

In what follows, we write $\boldsymbol{D}' \models s(\boldsymbol{D})$ to mean "database $\boldsymbol{D}'$ that has the same statistical profile $s(\boldsymbol{D})$", and $\boldsymbol{D}' \sim \mathcal{D}$ to mean "$\boldsymbol{D}'$ drawn from some database distribution $\mathcal{D}$". There are two types of cardinality estimators:

$$\hat{q} \approx \mathop{\mathbb{E}}_{\boldsymbol{D}' \sim \mathcal{D}} [|Q(\boldsymbol{D}')| \mid \boldsymbol{D}' \models s(\boldsymbol{D})] \qquad \text{average-case / model-based} \tag{1}$$

$$\hat{q} \approx \sup_{\boldsymbol{D}' \models s(\boldsymbol{D})} |Q(\boldsymbol{D}')| \qquad \text{worst-case / model-free} \tag{2}$$

The *average-case estimator* (1) relies on certain distributional assumptions about the data. The assumptions are modeled with a distribution $\mathcal{D}$ from which the data is drawn. The estimator aims to approximate the conditional expectation of $|Q(\boldsymbol{D}')|$ over all databases $\boldsymbol{D}'$ drawn from the distribution $\mathcal{D}$, conditioned on the fact that $\boldsymbol{D}'$ has the statistical profile $s(\boldsymbol{D})$. On the plus side, if $\mathcal{D}$ is a good model for the input data, then the estimator adapts well to the data, giving the query planner more accurate estimates, leading to better query plans. On the minus side, when the data does not conform on the assumptions baked into $\mathcal{D}$, one can end up with very bad query plans.

The *worst-case estimator* (2) approximates the worst-possible output size of query Q over all databases $\boldsymbol{D}'$ having the same statistical profile $s(\boldsymbol{D})$. (These are also called "pessimistic estimators" [11].) Two main advantages of worst-case estimators are that: (1) they are robust to outliers, heavy skews or corner cases do not affect the estimator and thus they help avoid query plans which explode in runtime under bad inputs, and (2) they can be used to guarantee memory budget during query evaluation. The main disadvantage is that, for a given dataset, the worst-case estimate may be far from the actual output size $|Q(\boldsymbol{D})|$, potentially leading to sub-optimal query plans. However, there are recent experimental research results showing that worst-case estimators can be quite effective on some benchmarks [11, 26].

Traditional approaches to cardinality estimation are all model-based. In the offline case (i.e. no sampling / learning), the main approach is stil the one proposed from the System-R days [54]. What we advocate for and present in this paper is model-free, designed to address the robustness concern of the traditional estimator.

The model-free estimation problem (2) is exactly the problem of estimating the worst-case output size of a join query. Studying this problem has led to a new class of join algorithms called *worst-case optimal join algorithms* [56, 47, 48]. For a certain class of statistical profile $s(\boldsymbol{D})$ (called "degree-constraints" in [6, 7]), the worst-case output size bound (2) is known to be deeply connected to important information theory questions [7, 5]). This is where our story begins: how do we solve the worst-case estimation problem (2) beyond the degree-constraints setup proposed in [7], taking into account, for example, histogram information?

The information-theoretic approach we present in this paper can be thought of as a "multiway cardinality estimator", to parallel the notion of "multiway-join operator" that worst-case optimal join algorithms are, as opposed to "binary-join cardinality estimator" which does not take into account constraints outside of the binary join being considered, which is also exactly why binary-join is not worst-case optimal!

The rest of this paper is organized as follows. Section 2 briefly presents the traditional cardinality estimation approach based on estimating the *selectivity*, and the notion of degree-constraints and polymatroid bounds. Section 3 presents an information-theoretic framework which begins with generalizing degree-constraints to the so-called "histogrammed frequency-moment constraints", and ends with an optimization problem for approximating (2) based on the entropy argument. Section 4 describes some of the main algorithmic and theoretical challenges we face while realizing the new approach. Section 5 concludes the paper.

2 Background

We use bold-face capital letters, such as $\boldsymbol{X}$, to denote tuples/sets of variables, capital letters such as X to denote a variable, bold-face lower-case letters, such as $\boldsymbol{x}$ to denote specific tuples in the domain $\mathsf{Dom}(\boldsymbol{X})$ of $\boldsymbol{X}$, and naturally non-bold-face letter x to denote a particular value in $\mathsf{Dom}(X)$.

2.1 Conjunctive Queries

We restrict our attention to estimating the output size of conjunctive queries, with a special emphasis on *full* conjunctive queries. We associate a *full conjunctive query* Q to a multi-hypergraph $\mathcal{H} := (\boldsymbol{V}, \mathcal{E})$, $\mathcal{E} \subseteq 2^{\boldsymbol{V}}$; the query is written as

$$Q(\boldsymbol{V}) \leftarrow \bigwedge_{\boldsymbol{F} \in \mathcal{E}} R_{\boldsymbol{F}}(\boldsymbol{F}), \tag{3}$$

with variables $\boldsymbol{V}$ and *atoms* $R_{\boldsymbol{F}}(\boldsymbol{F})$ for each $\boldsymbol{F} \in \mathcal{E}$. We also write $R_{\boldsymbol{F}}$ to avoid duplication. The atoms $R_{\boldsymbol{F}}$ represent either relational tables or built-in predicates whose columns are variables in $\boldsymbol{F}$.

▶ **Example 1.** *The following query corresponds to a triangle (hyper)graph, it is often called the "triangle query":*

$$Q_{\triangle}(A, B, C) \leftarrow E(A, B) \wedge E(A, C) \wedge E(B, C). \tag{4}$$

The relation E contains all edges of the graph that we want to count the number of triangles of.

Table 1 System-R-style selectivity estimation.

Predicate p	$s(p)$	note	default	assumption
$\neg p'$	$1 - s(p')$			
$p_1 \wedge p_2$	$s(p_1) \cdot s(p_2)$			independence
$A = c$	$1/d_A$	$d_A = \#$ of dist. vals	$\frac{1}{10}$	uniformity
$A > c$	$\frac{\max_A - c}{\max_A - \min_A}$	if known	$\frac{1}{3}$	uniformity
$c_1 < A < c_2$	$\frac{c_2 - c_1}{\max_A - \min_A}$	if known	$\frac{1}{4}$	uniformity
$R(A,B) \bowtie S(B,C)$	$\frac{1}{\max(d_B^R, d_B^S)}$	i.e. $\lvert R \bowtie S\rvert \approx \lvert R\rvert \cdot \lvert S\rvert \cdot s(\bowtie)$		uniformity
A IN L	$\min\{1/2, s(A = c)\lvert L\rvert\}$			
A IN Q	$\lvert Q\rvert/\lvert X\rvert$	X is cross-prod		

▶ **Example 2.** *Another example is the following query*

$$Q_+(A,B) \leftarrow R(A) \wedge S(B) \wedge A + B = 5 \tag{5}$$

Here, the predicate $A + B = 5$ is a built-in predicate, which is morally R_{AB} if we want to write it in the form (3)*.*

2.2 Traditional selectivity estimation

Most modern RDBMSs adopts a variant of System-R [53] cardinality estimation approach, which works as follows. Consider a (conjunctive) query which contains input relations and a collection of selection predicates. Without the predicates, the output size estimate is the product of input relation sizes. Each selection predicate reduces the estimate by a certain factor called the *selectivity* of the predicate. Table 1 summarizes how the selectivity of typical predicates are computed.

On the one hand, this approach has served us sufficiently well for the past 60 years or so, as evident by the commercial success. On the other hand, drawbacks of this approach are well-documented [38]. We list here a few key weaknesses:

- The approach is (probabilistic) model-based, but there does not seem to be a known theory for probabilistic guarantee regarding the quality of the estimate.
- The approach does not take into account all constraints at once, it is one predicate at a time, assuming independence. Hence, it is prone to under-estimation when the number of predicates involved is large.
- Is not entirely clear how to incorporate more known constraints into the estimator. The most obvious omission is in the fact that functional dependencies are *not* taken into account. In Example (2) above, for instance, our estimation approach will give a bound of $\min\{|R|, |S|\}$, while the traditional estimator approach likely gives $|R| \times |S|$.
- It is not clear how to estimate the output size of non-full conjunctive queries. (Of course, one can always take some trivial bound such as the cross-product of the distinct counts of the free variables.)
- The approach does not take into account histogram information when estimating join sizes. It does make use of the number of distinct values over the joined variable domain; however, in practice, histograms over the same attribute on different relations have mis-aligned boundaries.

2.3 Degree constraints

The notion of "degree constraints" was introduced in [7] to model a simple yet powerful form of statistical profiles $s(\boldsymbol{D})$. A *degree constraint* is a triple $(\boldsymbol{X}, \boldsymbol{Y}, N)$, where $\boldsymbol{X} \subsetneq \boldsymbol{Y} \subseteq \boldsymbol{V}$ and $N \in \mathbb{N}$. The relation $R_{\boldsymbol{F}}$ is said to *guard* the degree constraint $(\boldsymbol{X}, \boldsymbol{Y}, N)$ if $\boldsymbol{Y} \subseteq \boldsymbol{F}$ and

$$\max_{\boldsymbol{x}} |\pi_{\boldsymbol{Y}} \sigma_{\boldsymbol{X}=\boldsymbol{x}} R_{\boldsymbol{F}}| \leq N. \tag{6}$$

In plain language, the degree constraint states that: "in the relation $R_{\boldsymbol{F}}$, for every fixed binding $\boldsymbol{X} = \boldsymbol{x}$, there are at most N bindings $\boldsymbol{y}$ of $\boldsymbol{Y}$ for which $\boldsymbol{y}$ is in the projection of $R_{\boldsymbol{F}}$ onto the attributes $\boldsymbol{Y}$. Note that a given relation may guard multiple degree constraints.

Let DC denote a set of degree constraints. The input database $\boldsymbol{D}$ is said to *satisfy* DC if every constraint in DC has a guard, in which case we write $\boldsymbol{D} \models \mathsf{DC}$.

A *cardinality constraint* is an assertion of the form $|R_{\boldsymbol{F}}| \leq N$, for some $\boldsymbol{F} \in \mathcal{E}$; it is exactly the degree constraint $(\emptyset, \boldsymbol{F}, N)$ guarded by $R_{\boldsymbol{F}}$. A *functional dependency* $\boldsymbol{X} \to \boldsymbol{Y}$ is a degree constraint $(\boldsymbol{X}, \boldsymbol{Y}, 1)$. In particular, degree constraints strictly generalize both cardinality constraints and functional dependencies.

In the triangle query (4), suppose in addition to knowing that $|E| = N$ we also know that the out-degree of every vertex is bounded by D. Then this database (i.e. the graph G) satisfies the following degree constraints: $(\emptyset, \{u, v\}, N)$ for every pair $u, v \in [3]$. and $(\{1\}, \{1, 2\}, D)$ $(\{2\}, \{2, 3\}, D)$ and $(\{3\}, \{3, 1\}, D)$.

Our problem setting is general, where we are given a query of the form (3) and a set DC of degree constraints satisfied by the input database $\boldsymbol{D}$. The model-free cardinality estimation problem is to find a good upper bound of, or to determine exactly the quantity $\sup_{\boldsymbol{D} \models \mathsf{DC}} |Q(\boldsymbol{D})|$, the worst-case output size of the query given that the input satisfies the degree constraints. To describe the solution space, we need a detour to some classes of set functions.

2.4 Families of set functions

Let $n = |\boldsymbol{V}|$. A function $f : 2^{\boldsymbol{V}} \to \mathbb{R}_+$ is called a (non-negative) *set function* on $\boldsymbol{V}$. A set function f on $\boldsymbol{V}$ is *modular* if $f(\boldsymbol{S}) = \sum_{X \in \boldsymbol{S}} f(\{X\})$ for all $\boldsymbol{S} \subseteq \boldsymbol{V}$, is *monotone* if $f(\boldsymbol{X}) \leq f(\boldsymbol{Y})$ whenever $\boldsymbol{X} \subseteq \boldsymbol{Y}$, and is *sub-modular* if $f(\boldsymbol{X} \cup \boldsymbol{Y}) + f(\boldsymbol{X} \cap \boldsymbol{Y}) \leq f(\boldsymbol{X}) + f(\boldsymbol{Y})$ for all $\boldsymbol{X}, \boldsymbol{Y} \subseteq \boldsymbol{V}$. A function $h : 2^{\boldsymbol{V}} \to \mathbb{R}_+$ is said to be *entropic* if there is a joint distribution on $\boldsymbol{V}$ with entropy function H such that $h(\boldsymbol{S}) = H[\boldsymbol{S}]$ for all $\boldsymbol{S} \subseteq \boldsymbol{V}$.

Unless specified otherwise, we will only consider *non-negative* and *monotone* set functions f for which $f(\emptyset) = 0$; this assumption will be implicit in the entire paper. Furthermore, for $\boldsymbol{X} \subseteq \boldsymbol{Y}$, we will write $h(\boldsymbol{Y} \mid \boldsymbol{X}) := h(\boldsymbol{Y}) - h(\boldsymbol{X})$ for all our set functions h.

Let M_n and Γ_n denote the set of all (non-negative and monotone) modular and submodular set functions on $\boldsymbol{V}$, respectively. The set Γ_n is called the set of *polymatroidal functions*, or simply *polymatroids*. Let Γ_n^* denote the set of all entropic functions on n variables, and $\overline{\Gamma}_n^*$ denote its topological closure (in the Euclidean space, where we think of a polymatroid function $f : 2^{\boldsymbol{V}} \setminus \{\emptyset\} \to \mathbb{R}$ as a vector in $\mathbb{R}^{2^n - 1}$).

The notations $\Gamma_n, \Gamma_n^*, \overline{\Gamma}_n^*$ are standard in information theory. It is known [58] that Γ_n^* is a cone which is not topologically closed. And hence, when optimizing over this cone we take its topological closure $\overline{\Gamma}_n^*$, which is convex. It is easy to see that M_n and Γ_n are *polyhedral cones*. (Note that we can view them as either functions or vectors in $\mathbb{R}^{2^n - 1}$.)

There is another interesting class of set functions called *normal* functions [4, 6], defined as follows. For every $\boldsymbol{W} \subsetneq \boldsymbol{V}$, a *step function* $s_{\boldsymbol{W}} : 2^{\boldsymbol{V}} \to \mathbb{R}_+$ is defined by

$$s_{\boldsymbol{W}}(\boldsymbol{X}) = \begin{cases} 0 & \boldsymbol{X} \subseteq \boldsymbol{W} \\ 1 & \text{otherwise} \end{cases}$$

A function is normal if it is a non-negative linear combination of step functions. Let N_n denote the set of normal functions on $\boldsymbol{V}$.

As mentioned above, entropic functions satisfy non-negativity, monotonicity, and submodularity. Linear inequalities regarding entropic functions derived from these three properties are called *Shannon-type* inequalities. For a very long time, it was widely believed that Shannon-type inequalities form a complete set of linear inequalities satisfied by entropic functions, namely $\overline{\Gamma}_n^* = \Gamma_n$. This indeed holds for $n \leq 3$, for example. However, in 1998, in a breakthrough paper in information theory, Zhang and Yeung [59] presented a new inequality which cannot be inferred from Shannon-type inequalities. Their result proved that, $\overline{\Gamma}_n^* \subsetneq \Gamma_n$ for any $n \geq 4$.

The following inclusion chain can be found in a combination of [58, 4].

▶ **Theorem 3.** *The following chain of inclusion holds*

$$\mathsf{M}_n \subseteq \mathsf{N}_n \subseteq \Gamma_n^* \subseteq \overline{\Gamma}_n^* \subseteq \Gamma_n \tag{7}$$

When $n \geq 4$, all of the containments are strict.

2.5 Entropic and Polymatroid Bounds

In [7], we used families of set functions to describe answers to the worst-case cardinality estimation problem $\sup_{\boldsymbol{D} \models \mathsf{DC}} |Q(\boldsymbol{D})|$ This quantity is called the *worst-case output size* of the query, over databases satisfying the input degree constraints. Algorithms evaluating Q running in time $\tilde{O}(|\boldsymbol{D}| + \sup_{\boldsymbol{D} \models \mathsf{DC}} |Q(\boldsymbol{D})|)$ are called *worst-case optimal join algorithms* [47, 49, 56].

To obtain a bound in the general case, we employ the entropy argument, which is widely used in extremal combinatorics [34, 14, 50, 20], and in database theory [22, 6, 7]. The reader is referred to [47] for a brief historical account in relation to database theory.

Define the collection HDC of set functions satisfying the degree constraints DC:

$$\mathsf{HDC} := \{h \mid h : 2^{\boldsymbol{V}} \to \mathbb{R},\ h(\boldsymbol{Y}) - h(\boldsymbol{X}) \leq \log N, \qquad \forall (\boldsymbol{X}, \boldsymbol{Y}, N) \in \mathsf{DC}\}. \tag{8}$$

The entropy argument immediately gives the following result, first explicitly formulated in joint works with Abo Khamis and Suciu [6, 7]:

▶ **Theorem 4** (From [6, 7])**.** *Let Q be a conjunctive query and* DC *be a given set of degree constraints, then for any database $\boldsymbol{D}$ satisfying* DC*, we have*

$$\sup_{\boldsymbol{D} \models \mathsf{DC}} \log |Q(\boldsymbol{D})| = \max_{h \in \overline{\Gamma}_n^* \cap \mathsf{HDC}} h(\boldsymbol{V}) \qquad \textit{(entropic bound)} \tag{9}$$

$$\leq \max_{h \in \Gamma_n \cap \mathsf{HDC}} h(\boldsymbol{V}) \qquad \textit{(polymatroid bound)} \tag{10}$$

Furthermore, the entropic bound is asymptotically tight and the polymatroid bound is not.

3 An Information Theoretic Framework

This section first introduces a more powerful notion of constraints, enriching what can be stored in the profile $s(\boldsymbol{D})$. We explain how this type of constraints capture very well the kind of summary information that histograms store. Finally, we explain how to formulate a worst-case cardinality estimator subject to these constraints.

actor	movie	role
alice		
bob		
bob		
bob		
bob		
carol		
carol		

$$\begin{aligned} \boldsymbol{d}_{\mathsf{actor}}(\mathsf{alice}) &= 1 \\ \boldsymbol{d}_{\mathsf{actor}}(\mathsf{bob}) &= 4 \\ \boldsymbol{d}_{\mathsf{actor}}(\mathsf{carol}) &= 2 \\ \boldsymbol{d}_{\mathsf{actor}}(v) &= 0 \qquad\qquad v \notin \{\mathsf{alice}, \mathsf{bob}, \mathsf{carol}\} \end{aligned}$$

The (very small) profile $s(\boldsymbol{D})$ contains

- $F_0(\boldsymbol{d}_{\mathsf{actor}}) = \|\boldsymbol{d}_{\mathsf{actor}}\|_\infty = 4$ (*heavy hitter*)
- $F_\infty(\boldsymbol{d}_{\mathsf{actor}}) = \|\boldsymbol{d}_{\mathsf{actor}}\|_0 = 3$ (*distinct* counts)
- $F_1(\boldsymbol{d}_{\mathsf{actor}}) = \|\boldsymbol{d}_{\mathsf{actor}}\|_1 = 7 = |R|$ (base-table *cardinality*)
- $F_2(\boldsymbol{d}_{\mathsf{actor}}) = \|\boldsymbol{d}_{\mathsf{actor}}\|_2^2 = 21$ (2nd moment)

Figure 2 An example of degree-norm constraints.

3.1 Frequency-moment constraints

3.1.1 Motivation

To motivate the notion of frequency-moment constraints, let us consider an example shown in Figure 2. Let's say we have a table $R(\mathsf{actor}, \mathsf{movie}, \mathsf{role})$ and we would like to compute a small statistical profile (that goes into $s(\boldsymbol{D})$) of this table. We want to be able to capture as much of the joint distribution over three variables actor, movie, and role using as little space as possible.

One possible representation is to look at each of the marginal distributions over individual variables. On the variable actor, the marginal distribution is summarized with a *frequency vector* $\boldsymbol{d}_{\mathsf{actor}}$ which counts, for each actor, the number of times the actor occurs in the table. This marginal mass vector is typically too large to be part of the profile $s(\boldsymbol{D})$. Instead, the idea of frequency-moment constraints is to include in $s(\boldsymbol{D})$ some frequency-moment [8] of this vector: $F_\ell(\boldsymbol{d}_{\mathsf{actor}})$, for $\ell \in \{0, 1, 2, \infty\}$. The *frequency-moment* F_ℓ of a vector $\boldsymbol{v}$ is defined by

$$F_\ell(\boldsymbol{v}) := \begin{cases} \|\boldsymbol{v}\|_\ell & \ell \in \{0, 1, +\infty\} \\ \|\boldsymbol{v}\|_\ell^\ell & \ell \notin \{0, 1, +\infty\}. \end{cases} \tag{11}$$

Theoretically, we are certainly free to pick ℓ-moments for values of ℓ beyond $\{0, 1, 2, \infty\}$, but practically they are not very meaningful. As shown in Figure 2, the 0-, 1-, and ∞-moments capture commonly used statistics in RDBMSs: heavy hitters, distinct value counts, and base-table cardinalities.

3.1.2 Formal definition

Let R be a relation, and $\boldsymbol{X}, \boldsymbol{Y}$ be subsets of attributes of the relation. Define the *conditional frequency vector* $\boldsymbol{d}_{\boldsymbol{Y}|\boldsymbol{X}}$ to be

$$\boldsymbol{d}^R_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{x}) = |\pi_{\boldsymbol{Y}} \sigma_{\boldsymbol{X}=\boldsymbol{x}} R| \tag{12}$$

When R is clear from context, we drop the subscript R to reduce cluttering. Note that $\boldsymbol{X}$ can be empty, where $\boldsymbol{d}^R_{\boldsymbol{Y}|\emptyset}() = |\pi_{\boldsymbol{Y}}(R)|$ counts the number of distinct $\boldsymbol{Y}$-tuples in R.

▶ **Definition 5** (Frequency-moment constraint)**.** *A* frequency-moment constraint *(or just* FM-constraints *for short) is is a quintuple* $(\boldsymbol{X}, \boldsymbol{Y}, N, \ell, R)$*, where* $\boldsymbol{X} \subseteq \boldsymbol{Y}$ *are sets of variables,* $N \in \mathbb{R}_+$*, and* $\ell \in [0, +\infty]$ *is a nonnegative real number or infinity.* R *is an input relation. The constraint states that*

$$F_\ell(\boldsymbol{d}^R_{Y|X}) \le N \tag{13}$$

The values of ℓ that are most meaningful in practice are $\{0, 1, 2, +\infty\}$.

Note the following fact: for the same relation R, we have

$$F_0(\boldsymbol{d}_{\boldsymbol{Y}|\boldsymbol{X}}) = F_\infty(\boldsymbol{d}_{\boldsymbol{X}|\emptyset}) \qquad F_1(\boldsymbol{d}_{\boldsymbol{Y}|\boldsymbol{X}}) = F_\infty(\boldsymbol{d}_{\boldsymbol{X}\cup\boldsymbol{Y}|\emptyset}) \tag{14}$$

In particular, adding the ability to measure 1- or 0-moments does not move us beyond the degree constraints setting of ∞-frequency-moments. This fact will change when we use FM-constraints in the context of histograms, as presented in the next section.

In the obvious way, FM-constraints capture profile information typically used in RDBMSs, such as functional dependencies, base-table cardinalities, distinct value counts, and heavy hitters. It is a strict generalization of degree constraints.

3.1.3 Related recent works

Independent of our work, there are a couple of recent works which dealt with degree sequences and their norms.

Jayaraman, Ropell and Rudra [32] considered the join problem where the input database contains arity-2 relations, each of which has a degree vector some of whose norms are given as input to the join computation problem. They derived a worst-case optimal join algorithm under this input.

Deeds, Suciu, Balazinska, and Cai [17] considered the problem setting where entire degree vectors $\boldsymbol{d}^R_{\boldsymbol{Y}|X}$ are given, along with maximum tuple multiplicities. They derived novel bounds for the output size given this information.

The setup and results of both papers are orthogonal to what is presented in this talk.

3.2 Histograms

The frequency-moment constraints are powerful building blocks for summarizing the data in a database $\boldsymbol{D}$. To increase the granularity of the statistical profile $s(\boldsymbol{D})$, we partition the data and have frequency-moment constraints for each part of the data space. This is the idea behind histograms [2, 29]. Given a relation $R_{\boldsymbol{Y}}$ and a set $\boldsymbol{X} \subset \boldsymbol{Y}$ of its attributes, to build an $\boldsymbol{X}$*-histogram*, we partition the active domain of $\boldsymbol{X}$ into some k parts:

$$\mathsf{Dom}(\boldsymbol{X}) := \prod_{X\in\boldsymbol{X}} \mathsf{Dom}(X) = B_1 \cup B_2 \cup \cdots \cup B_k.$$

The B_is are called "buckets". Let $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$. For each bucket $B \in \mathcal{B}$, the B-conditional frequency vector $\boldsymbol{d}^R_{\boldsymbol{Y}|\boldsymbol{X}\in B}$ is defined by:

$$\boldsymbol{d}^R_{\boldsymbol{Y}|\boldsymbol{X}\in B}(\boldsymbol{x}) = |\pi_{\boldsymbol{Y}}\sigma_{\boldsymbol{X}=\boldsymbol{x}}R| \qquad \boldsymbol{x} \in B \tag{15}$$

As usual, we drop R and write $\boldsymbol{d}_{\boldsymbol{Y}|\boldsymbol{X}\in B}$ when R is clear from context.

If $|\boldsymbol{X}| = 1$, then the histogram is a 1D-histogram, otherwise it is a multidimensional histogram. In typical RDBMSs, there are about $k \approx 200$ buckets (e.g. MS SQL Server). Furthermore, the top 10 or so heavy hitters are each in a bucket by themselves. For 1D-histograms, the bucketization is done via *equi-depth* partitioning[29]. Multi-dimensional histograms are algorithically more complicated and require more space to store [46, 45].

▶ **Definition 6.** *A* histogrammed FM-constraint *(or HFM-constraint for short) is a tuple* $(\mathcal{B}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{c}, \ell, R)$*, where* $\boldsymbol{X} \subseteq \boldsymbol{Y}$ *are sets of variables,* $\mathcal{B}$ *is a partition of* $\mathsf{Dom}(\boldsymbol{X})$*,* $\boldsymbol{c} = (c_B)_{B\in\mathcal{B}}$ *is a vector of real numbers, and* $\ell \in [0, +\infty]$*. The constraint states that*

$$F_\ell(\boldsymbol{d}^R_{\boldsymbol{Y}|\boldsymbol{X}\in B}) \le c_B \qquad \forall B \in \mathcal{B} \tag{16}$$

Unlike in the histogram-free case, we can no longer use the ∞-moments to capture the 0- and 1-moments. Each moment is its own useful statistics.

3.3 A model-free cardinality estimator under HFM-constraints

This section explains how in an on-going work, in collaboration with Keshavarz and Nguyen [35], we were able to generalize Theorem 4 to the case when the input constraints are HFM-constraints. For the sake of clarity and to simplify the exposition, we will restrict the HFM-constraints to only 1D-histogram constraints.[2]

3.3.1 A highly simplified example

To illustrate the main ideas, we start with an example where the query is a simple join between two relations

$$Q(X, Y, Z) = R(X, Y) \wedge S(Y, Z).$$

We further assume that there are HFM-constraints on both R and S, where the bucketization on Y is *identical* on both R and S. In particular, suppose the input HFM-constraints are of the form:

- $(\mathcal{B}, Y, XY, \boldsymbol{r}, \infty, R)$, where $\boldsymbol{r} = (r_B)_{B\in\mathcal{B}}$
- $(\mathcal{B}, Y, XY, \boldsymbol{c}, 0, R)$, where $\boldsymbol{c} = (c_B)_{B\in\mathcal{B}}$
- $(\mathcal{B}, Y, YZ, \boldsymbol{s}, \infty, S)$, where $\boldsymbol{s} = (s_B)_{B\in\mathcal{B}}$

More concretely, the constraints state the following, for *every* $B \in \mathcal{B}$:

- given any $y \in B$, we have $|\pi_X \sigma_{Y=y} R| \le r_B$
- $|\sigma_{Y\in B} \pi_Y R| \le c_B$
- given any $y \in B$, we have $|\pi_X \sigma_{Y=y} S| \le s_B$

We now apply the entropy argument to upper-bound $|Q|$. The starting point is the traditional entropy argument. Fix a particular (but arbitrary) input, including relations R and S. Consider the uniform distribution on (X, Y, Z) chosen from the join $R(X, Y) \wedge S(Y, Z)$. (Note that we do not assume anything about input distribution; the uniformity considered here is only for mathematical reasoning purposes.) Next, we depart from the entropy argument used to prove the likes of Theorem 4. We add one more random variable to the joint distribution. Let $J \in \mathcal{B}$ be the categorical variable, where $J = B$ iff $Y \in B$. Define

$$p_B := \mathsf{Pr}[J = B] \qquad \boldsymbol{p} := (p_B)_{B\in\mathcal{B}} \tag{17}$$

Consider the joint distribution on (X, Y, Z, J). Let h be its entropy function. Then, $h \in \overline{\Gamma}^*_4$; and the following holds:

$$\log|Q| = h(XYZ) \qquad h(J \mid Y) = 0 \qquad h(J) = -\sum_i p_i \log p_i \qquad \|\boldsymbol{p}\|_1 = 1 \qquad \boldsymbol{p} \ge \boldsymbol{0} \tag{18}$$

[2] This is the typical case in all existing RDBMSs: by default multidimensional histograms are not built. Sometimes they are built on-demand, but they are not used in estimating the join sizes, only to estimate selectivities of filter conditions on the base-tables.

Furthermore, the following inequalities hold, for every $B \in \mathcal{B}$:

$$h(X \mid J = B) \le \log r_B \tag{19}$$
$$h(Y \mid J = B) \le \log c_B \tag{20}$$
$$h(Z \mid J = B) \le \log s_B. \tag{21}$$

We simplify the constraints above by aggregating them:

$$h(X \mid J) = \sum_B h(X \mid J = B) \cdot p_B \le \langle \log \boldsymbol{r}, \boldsymbol{p} \rangle \tag{22}$$
$$h(Y \mid J) = \sum_B h(Y \mid J = B) \cdot p_B \le \langle \log \boldsymbol{d}, \boldsymbol{p} \rangle \tag{23}$$
$$h(Z \mid J) = \sum_B h(Z \mid J = B) \cdot p_B \le \langle \log \boldsymbol{s}, \boldsymbol{p} \rangle \tag{24}$$

Overall, we have the following optimization problem, which is our worst-case cardinality estimator for the example input:

$$\begin{aligned}
\max \quad & h(XYZ) && (25)\\
\text{s.t.} \quad & h(Y \mid J) \le \langle \boldsymbol{p}, \lg \boldsymbol{c} \rangle && (26)\\
& h(X \mid J) \le \langle \boldsymbol{p}, \lg \boldsymbol{r} \rangle && (27)\\
& h(Z \mid J) \le \langle \boldsymbol{p}, \lg \boldsymbol{s} \rangle && (28)\\
& h \in \overline{\Gamma}_4^* && (29)\\
& \boldsymbol{p} \ge 0, && (30)\\
& h(J) = -\langle \boldsymbol{p}, \lg \boldsymbol{p} \rangle && (31)\\
& h(J \mid Y) = 0 && (32)\\
& \|\boldsymbol{p}\|_1 = 1. && (33)
\end{aligned}$$

For this problem to be solvable, we replace $\overline{\Gamma}_4^*$ with Γ_4, which contains all the Shannon-type inequalities on h.

To show that the above optimization problem makes sense and is non-trivial, we now show that the estimator matches a combinatorial bound. We do so by applying only the constraints from the above optimization problem (with $\overline{\Gamma}_4^*$ replaced by Γ_4) to derive a bound on $|Q|$:

$$\begin{aligned}
\lg |Q| &= h(XYZ) && (34)\\
(\text{since } h(JY] = h(Y]) &= h(XYZJ) = h(XYZ|J) + h(J) && (35)\\
(\text{since } h \in \Gamma_4) &\le h(X|J) + h(Y|J) + h(Z|J) + h(J) && (36)\\
&\le \sum_{B \in \mathcal{B}} (\lg r_B + \lg c_B + \lg s_B - \lg p_B)) \cdot p_B && (37)\\
&= \sum_{B \in \mathcal{B}} (\lg(r_B c_B s_B / p_B)) \cdot p_B && (38)\\
(\text{Jensen}) &\le \lg \left(\sum_B r_B c_B s_B \right). && (39)
\end{aligned}$$

We arrive at the bound

$$|Q| \le \sum_{B \in \mathcal{B}} r_B c_B s_B \tag{40}$$

which is *exactly* what we would expect from the input conditions; furthermore, it is easy to see that the bound is tight!

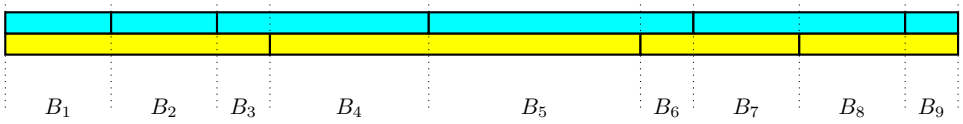


Figure 3 When boundaries of the blue-histograms and yellow histograms do not align.

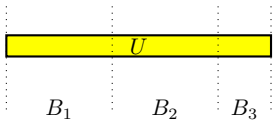


Figure 4 Refined partitioning of a yellow interval.

3.3.2 When histogram boundaries do not align

We now remove the unrealistic assumption that the histogram boundaries on R and S align perfectly. We assume the input HFM-constraints are

- $(\mathcal{B}', Y, XY, \boldsymbol{r}, \infty, R)$, where $\boldsymbol{r} = (r_{B'})_{B' \in \mathcal{B}'}$
- $(\mathcal{B}', Y, XY, \boldsymbol{c}, 0, R)$, where $\boldsymbol{c} = (c_{B'})_{B' \in \mathcal{B}'}$
- $(\mathcal{B}'', Y, YZ, \boldsymbol{s}, \infty, S)$, where $\boldsymbol{s} = (s_{B''})_{B'' \in \mathcal{B}''}$

The natural idea is to compute a refined partition $\mathcal{B}$ of $\mathsf{Dom}(Y)$, from intersecting the partitions $\mathcal{B}'$ and $\mathcal{B}''$, as shown in Figure 3. The issue is that we do not know the numbers r_B, c_B, and s_B; hence, we make them variables, and write down extra constraints to relate them to $r_{B'}$, $c_{B'}$, and $s_{B''}$.

The extra constraints depend on the norm-ℓ; for instance, suppose an interval $U \in \mathcal{B}'$ is refined into $B_1 \cup B_2 \cup B_3$ as shown in Figure 4, then we have two extra constraints:

$$c_{B_1} + c_{B_2} + c_{B_3} = c_U \tag{41}$$

$$\max\{r_{B_1}, r_{B_2}, r_{B_3}\} = r_U. \tag{42}$$

3.3.3 The general case

The above examples can be generalized as follows. Consider a collection HFM of HFM-constraints, for a query Q over variables $\boldsymbol{V}$. Recall that HFM-constraints are a strict super set of FM-constraints, which is a strict superset of DC-constraints, which is a strict superset of FD-constraints and cardinality constraints. An HFM-constraint $(\mathcal{B}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{c}, \ell, R)$ is an FM-constraint if $\mathcal{B} = \{\mathsf{Dom}(X)\}$, and an FM-constraint is a DC-constraint if $\ell = \infty$. An HFM-constraint is called *proper* if $|\mathcal{B}| > 1$ (i.e. it is a legitimate partition of $\mathsf{Dom}(\boldsymbol{X})$). Thanks to (14), if $\ell \in \{0, 1, \infty\}$, then we can assume that all non-proper HFM-constraints are DC-constraints. The size $|\boldsymbol{X}|$ is called the *dimensionality* of the HFM-constraint.

▶ **Definition 7.** *We call a collection $\mathcal{C}$ of HFM-constraints* simple *if the following conditions are met:*

- *Every constraint in* HFM *has* $\ell \in \{0, 1, +\infty\}$
- *Every proper HFM-constraint in* HFM *is one-dimensional (i.e.* $|\boldsymbol{X}| = 1$*)*

The idea to parallel Theorem 4 is to write down a set HC of all constraints, and formulate an optimization problem which mirrors the above examples and that of Theorem 4. We will assume that the input collection $\mathcal{C}$ of HFM-constraints are simple.

To start describing HC, we begin by assuming the "aligned boundary" case for all proper HFM-constraints. In particular, if there was at least one proper HFM-constraint $(\mathcal{B}, X, \boldsymbol{Y}, \boldsymbol{c}, \ell, R)$ on X, then *all* other constraints of the form $(\mathcal{B}', X, \boldsymbol{Y}', \boldsymbol{c}', \ell', R')$ on X, then *all* other constraints of the form must have $\mathcal{B}' = \mathcal{B}$.

For each such X, we add a new categorical variable J (to the joint distribution over $\boldsymbol{V}$), and a new vector $\boldsymbol{p} = (p_B)_{B\in\mathcal{B}}$, with the following constraints to HC:

$$h(J) = -\langle \boldsymbol{p}, \lg \boldsymbol{p}\rangle \qquad h(J \mid X) = 0 \qquad \|\boldsymbol{p}\|_1 = 1 \qquad \boldsymbol{p} \geq \boldsymbol{0} \tag{43}$$

Next, for each HFM-constraint $(\mathcal{B}, X, \boldsymbol{Y}, \boldsymbol{c}, \ell, R)$, we add the following constraints to HC:

$$\text{if } \ell = 0 \qquad h(X|J) \leq \langle \boldsymbol{p}, \lg \boldsymbol{d}\rangle \tag{44}$$

$$\text{if } \ell = 1 \qquad h(X\boldsymbol{Y}|J) \leq \langle \boldsymbol{p}, \lg \boldsymbol{d}\rangle \tag{45}$$

$$\text{if } \ell = \infty \qquad h(\boldsymbol{Y}|J) \leq \langle \boldsymbol{p}, \lg \boldsymbol{d}\rangle \tag{46}$$

The constraints for $\ell \in \{0, \infty\}$ were already explained in the example above (see (22), (23), (24)). The constraint for $\ell = 1$ also follows the same reasoning; the only difference is that the F_1-frequency moment counts the number of $(X, \boldsymbol{Y})$ tuples, and hence the bound is on $h(X\boldsymbol{Y}|J)$.

Now, when the boundaries of all the bucketizations $\mathcal{B}$ on the same variable X are *not* aligned, we do the following.

- Create the finest partition $\mathcal{B}$ of $\mathsf{Dom}(X)$ from taking the intersections of all the input bucketizations $\mathcal{B}'$ on $\mathsf{Dom}(X)$.
- For each input HFM-constraint $(\mathcal{B}', X, \boldsymbol{Y}, \boldsymbol{c}', \ell, R)$, create a new HFM-constraint $(\mathcal{B}, X, \boldsymbol{Y}, \boldsymbol{c}, \ell, R)$, where $\boldsymbol{c}$ are variables
- Add the following constraints on the quantities $\boldsymbol{c}$, depending on ℓ. Since $\mathcal{B}$ is a finer partition than $\mathcal{B}'$, every interval $U \in \mathcal{B}'$ is a union of some intervals $U = B_1 \cup \cdots \cup B_q$ in $\mathcal{B}$. We relate $c_{B_1}, \ldots, c_{B_q}$ to c'_U as follows.

$$\text{if } \ell \in \{0, 1\} \qquad c_{B_1} + \cdots + c_{B_k} \leq c'_U \tag{47}$$

$$\text{if } \ell = \infty \qquad c_{B_i} \leq c'_U \qquad \forall i \tag{48}$$

Let m denote the number of variables X for which there is a bucketization from $\mathcal{C}$ from, then the joint distribution we considered is on $n + m$ variables: $n = |\boldsymbol{V}|$, and there is one variable J for each such X. Thus, $h \in \overline{\Gamma}^*_{n+m}$. In addition, we have variables $\boldsymbol{p}$ for each X, and new variables $\boldsymbol{c}$ for each input HFM-constraint on which the bucketizations do not align. Together, the unknowns involve $(h, \boldsymbol{P}, \boldsymbol{C})$ where $\boldsymbol{P}$ collects all the unknowns $\boldsymbol{p}$, and $\boldsymbol{C}$ collects all the unknown $\boldsymbol{c}$. Let HC denote the list of all constraints. Then, we have the following.

▶ **Theorem 8** (From [35]). *Let Q be a conjunctive query and $\mathcal{C}$ be a given set of simple HFM-constraints, then for any database $\boldsymbol{D}$ satisfying $\mathcal{C}$, we have*

$$\sup_{\boldsymbol{D}\models\mathcal{C}} \log|Q(\boldsymbol{D})| \leq \max\{h(\boldsymbol{V}) \mid h \in \overline{\Gamma}^*_{n+m}, (h, \boldsymbol{P}, \boldsymbol{C}) \in \mathsf{HC}\} \quad \textit{(entropic bound)} \tag{49}$$

$$\leq \max\{h(\boldsymbol{V}) \mid h \in \Gamma_{n+m}, (h, \boldsymbol{P}, \boldsymbol{C}) \in \mathsf{HC}\} \quad \textit{(polymatroid bound)} \tag{50}$$

The (generalized) polymatroid bound (50) is our model-free estimator. It is possible to relax the simplicity assumption on $\mathcal{C}$, but the description of the abound will be much more involved. For the sake of clarify, we refrain from doing so here.

4 Research Questions

The approach we sketched in the previous section raises some very interesting and challenging research problems, with deep connections to information theory, linear programming, combinatorial optimization, and statistical estimation. This section outlines some research questions arising from our framework. We taxonomize the research questions in three broad categories:

- The first category involves questions surrounding the computability of the entropic bounds.
- The second category involves questions on how to compute the polymatroid bound efficiently.
- The third category aims to capture questions where probabilistic guarantees are taken into account, getting data-type specific information and the 2nd-moment ($\ell = 2$) constraints involved in the model.

4.1 Computability and information inequality

In order to compute the best (worst-case) cardinality estimate, we want to compute the entropic bound. For simplicity, let's start with the entropic bound (9) under degree constraints only. The constraints in HDC are linear constraints, and $\overline{\Gamma}_n^*$ is known to be a closed convex cone. Hence, the entropic bound is a *conic programming* problem [10] of the form:

$$\begin{aligned} \min \quad & \langle \boldsymbol{c}, \boldsymbol{h} \rangle && (51)\\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{h} \le \boldsymbol{b} && (52)\\ & \boldsymbol{h} \in \overline{\Gamma}_n^*, && (53) \end{aligned}$$

where $c_{\boldsymbol{V}} = -1$ and $c_{\boldsymbol{X}} = 0$ for $\boldsymbol{X} \subset \boldsymbol{V}$. The inequalities in $\boldsymbol{A}\boldsymbol{h} \le \boldsymbol{b}$ come from the degree constraints: $h(\boldsymbol{Y}) - h(\boldsymbol{X}) \le N$. We write down a particular Lagrangian dual problem. To do so, associate dual variables $\boldsymbol{\delta}$ to the inequalities $\boldsymbol{A}\boldsymbol{h} \le \boldsymbol{b}$. The Lagrangian is

$$\mathcal{L}(\boldsymbol{\delta}) = \inf_{\boldsymbol{h} \in \overline{\Gamma}_n^*} \langle \boldsymbol{c}, \boldsymbol{h} \rangle + \langle \boldsymbol{A}\boldsymbol{h} - \boldsymbol{b}, \boldsymbol{\delta} \rangle = -\langle \boldsymbol{b}, \boldsymbol{\delta} \rangle + \inf_{\boldsymbol{h} \in \overline{\Gamma}_n^*} \langle \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta}, \boldsymbol{h} \rangle \tag{54}$$

Let $(\overline{\Gamma}_n^*)^*$ denote the *dual cone* of the cone $\overline{\Gamma}_n^*$.

- If $\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta} \in \overline{\Gamma}_n^*$, then $\langle \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta}, \boldsymbol{h} \rangle \ge 0$ and thus $\inf_{\boldsymbol{h} \in \overline{\Gamma}_n^*} \langle \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta}, \boldsymbol{h} \rangle = 0$.
- If $\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta} \notin \overline{\Gamma}_n^*$, then $\inf_{\boldsymbol{h} \in \overline{\Gamma}_n^*} \langle \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta}, \boldsymbol{h} \rangle = -\infty$.

Since the Lagrangian dual problem is to maximize $\mathcal{L}(\boldsymbol{\delta})$ subject to $\boldsymbol{\delta} \ge \boldsymbol{0}$, solving the above conic programming problem is essentially equivalent to solving the following dual:

$$\begin{aligned} \min \quad & \langle \boldsymbol{b}, \boldsymbol{\delta} \rangle && (55)\\ \text{s.t.} \quad & \boldsymbol{\delta} \ge \boldsymbol{0} && (56)\\ & \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta} \in (\overline{\Gamma}_n^*)^*. && (57) \end{aligned}$$

There is evidence that the dual problem (55) may not be decidable. Even checking for feasibility of a given solution seems hard. The reason is as follows. The statement $\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta} \in (\overline{\Gamma}_n^*)^*$ is equivalent to

$$\langle \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\delta}, \boldsymbol{h} \rangle = \langle \boldsymbol{c}, \boldsymbol{h} \rangle + \langle \boldsymbol{A}\boldsymbol{h}, \boldsymbol{\delta} \rangle \ge 0 \qquad \forall h \in \overline{\Gamma}_n^*.$$

In particular, this is saying that the inequality

$$h(\boldsymbol{V}) \le \sum_{(X,Y) \in \mathsf{DC}} \delta_{Y|X} h(Y|X) \tag{58}$$

is a valid *information inequality*, i.e. an inequality that holds for all almost entropic functions. In joint work with Abo Khamis, Kolaitis, and Suciu [4], we studied this class of problems (of deciding the validity of information inequalities, and their generalization). While the

(un)decidability of these bounds are open, we were able to put them on the arithmetic hierarchy. Studying these inequalities are also closely related to (the decidability of) the problem of query containment under bag-semantic [4].

On the plus side, it was known [7] that the entropic bound under degree constraints is asymptotically tight! It is open, however, whether the generalization of the bound under HFM-constraints (49) is tight or not.

4.2 Computational complexity of the polymatroid bounds

Our next best hope is thus put on the polymatroid bound (10) and its histogrammed counterpart (50). These are optimization problems on an exponential number of variables and constraints.

Consider the simpler bound (10), under input DC constraints. On the negative news side, it is known [7] that the bound is not tight in general, namely there is a gap between the entropic bound (9) and the polymatroid bound on some input instances. In fact, one can construct a family of input instances for which the gap-ratio goes to infinity. Furthermore, as mentioned in [47], the exact computational complexity of computing the polymatroid bound (10) is open.

What gives us hope that the bound is computably tractable to begin with? After all, the linear program has an exponential number of variables and constraints. This is when some positive news emerge. We know that, under certain assumptions about the input degree constraints, we know that the polymatroid bound is not only computable in polynomial time, but also is tight (i.e. it is equal to the entropic bound:

- If DC contains only cardinality constraints, then we can show that [6] the polymatroid bound is exactly equal to the AGM bound [9]. One way to prove this is to use Lovasz "modularization" technique to show that one can replace polymatroids by *modular* polymatroids in the optimization problem while retaining the same objective value. The dual of the modular polymatroid optimization problem is exactly the AGM bound.
- There is one simple further relaxation we can make: if in addition to cardinality constraints, we have *simple* FDs, then the bound is also tight an computable in PTime. This fact was observed in [22] and generalized in [6] in terms of the lattice of FD closures. In particular, it was shown in [6] that if the FD-closure lattice is *distributive* [55], then the bound is tight and computable in PTime. (Simple FDs implies distributive FD-closure lattice.)
- Another case when the bound is tight and PTime-computable is when the set DC of input degree constraints is acyclic [47]. One can use this fact in another way, in order to obtain an upper-approximation of the bound: find a minimal subset of DC that is acyclic and use that as the approximation.
- Finally, recently in [28] we showed that if all degree constraints are *simple* then the bound is tight and PTime-computable. A degree constraint is simple if it is of the form (X, Y, N) with $|X| \leq 1$. In particular, all cardinality constraints are simple, all simple FDs are simple, and in addition we can also have proper degree constraints with $|X| = 1$. The fact that this bound is tight was shown in [4] where we showed that the polymatroids can be replaced by *normal* polymatroids in this case. Thanks to (7), this proves tightness of the bound. However, it still requires an exponential-sized description to describe the normal polymatroids N_n. In [28] we proved PTime-computability by characterizing the optimal solution using network flow analysis. We also proved that the bound is tight using a different strategy than what was used in [28]. Surprisingly, if the input DC are not (necessarily) simple, then computing the normal-polymatroid bound is NP-hard [28].

Next, consider the bound (50) under HFM-constraints. This is a concave maximization problem, which can be numerically solved [10]. An interesting research question is to find good upper-approximation to this bound that can be computed efficiently. Following the Jensen inequality strategy presented in Section 3.3.1, we can eliminate the extra variables J and $\boldsymbol{p}$; however, we are still studying how much we lose by this approximation [35].

4.3 Probabilistic guarantees

Thus far, we have not made use of the 2nd frequency-moment ($\ell = 2$) in the information theoretic framework. An interesting open problem is to devise a natural way to incorporate ℓ-moments for $\ell \notin \{0, 1, +\infty\}$ – especially $\ell = 2$.

One possible way to make use of $\ell = 2$ is to start incorporating probabilistic guarantees into our estimator (instead of a guaranteed upper-bound with probability 1). For example, consider the $R(\mathsf{actor}, \mathsf{movie}, \mathsf{role})$ relation and suppose we have $F_2(\boldsymbol{d}_{\mathsf{actor}})$ as part of $s(\boldsymbol{D})$. Suppose the query is $R(\mathsf{actor}, \mathsf{movie}, \mathsf{role}) \wedge \mathsf{actor} =$ "KevinBacon".

- In the System-R approach, the estimate will be $\frac{|R|}{F_0(\boldsymbol{d}_{\mathsf{actor}})}$. This is the average actor-degree, which would be an under estimate if "Kevin Bacon" was a heavy hitter in the table.
- In the information theoretic approach, before taking $\ell = 2$ into account, our estimate would have been $F_\infty(\boldsymbol{d}_{\mathsf{actor}})$, which is good for a heavy hitter, but would be an over estimate if "Kevin Bacon" is *not* a heavy hitter.
- Having $F_2(\boldsymbol{d}_{\mathsf{actor}})$ allows us to take a probabilistic compromise. Recall Cantelli's inequality, which says that, for *any* random variable X

$$\mathsf{Pr}\left[X \geq \mathbb{E}[X] + \lambda\right] \leq \frac{\mathsf{Var}[X]}{\mathsf{Var}[X] + \lambda^2}$$

Let X be the frequency (degree) of "Kevin Bacon". We do not know the distribution of X. To be as model-free as possible, we follow the *maximum-entropy principle* [33] and assume "Kevin Bacon" is uniformly distributed among all actors. Then,

$$\mathbb{E}[X] = \frac{|R|}{F_0(\boldsymbol{d}_{\mathsf{actor}})} \tag{59}$$

$$\mathsf{Var}[X] = \frac{F_2(\boldsymbol{d}_{\mathsf{actor}})}{F_0(\boldsymbol{d}_{\mathsf{actor}})} - \mathbb{E}[X]^2. \tag{60}$$

From these quantities and Cantelli's inequality, we can strike a balance between the traditional approach and our approach: we can guarantee that our upper-bound estimate is correct with a certain probabilistic threshold.

The idea of applying the maximum entropy principle has been used successfully in selectivity estimation [43]. It can also be used to deal with other types of information one typically record in database catalogs: the (non-frequency) moments of continuous variables. For example, the System-R estimator for the predicate $A > c$ is $\frac{\max_A - c}{\max_A - \min_A}$, as shown in Table 1. This assumes a uniform distribution over the interval $[\min_A, \max_A]$. However, if we also collect the empirical mean and variance of the variable, then assuming uniformity may not make sense for these given statistics. Instead, following the maximum entropy principle, we should fit an exponential family distribution to model and bound this estimate. Then, Cantelli or Chebyshev inequality can be used to give the probabilistic guarantee at the desired level.

5 Conclusions

We presented a recent effort at RelationalAI to formulate and devise a solution to the classic cardinality estimation problem in query optimization. Our approach aims to be model-free, or as model-free as possible, in order to avoid well-documented shortcomings of the traditional selectivity estimation approach. Our formulation is only for the offline case: no sampling nor learning was incorporated.

The approach is promising, as a variant of it is working well in production. There remain highly interesting and non-trivial open questions, as presented. We sincerely hope this presentation inspires more database theorists to study the problem. There are enough deep connections to information theory, algorithms, optimization, and statistics for long-term research programs.

References

1 MS SQL server documentation, 2021. URL: https://docs.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver15.

2 Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *VLDB J.*, 24(4):557–581, 2015. doi:10.1007/s00778-015-0389-y.

3 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

4 Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. Bag query containment and information theory. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 95–112. ACM, 2020. doi:10.1145/3375395.3387645.

5 Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. Decision problems in information theory. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 106:1–106:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.106.

6 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 327–342. ACM, 2016. doi:10.1145/2902251.2902289.

7 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 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 429–444. ACM, 2017. doi:10.1145/3034786.3056105.

8 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 20–29. ACM, 1996. doi:10.1145/237814.237823.

9 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.43.

10 Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, 2004. doi:10.1017/CBO9780511804441.

11 Walter Cai, Magdalena Balazinska, and Dan Suciu. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 18–35. ACM, 2019. doi:10.1145/3299869.3319894.

12 Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In Carsten Lutz and Jean Christoph Jung, editors, *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*, volume 155 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICDT.2020.7.

13 Zhiyuan Chen, Flip Korn, Nick Koudas, and S. Muthukrishnan. Selectivity estimation for boolean queries. In Victor Vianu and Georg Gottlob, editors, *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, pages 216–225. ACM, 2000. doi:10.1145/335168.335225.

14 F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer. Some intersection theorems for ordered sets and graphs. *J. Combin. Theory Ser. A*, 43(1):23–37, 1986. doi:10.1016/0097-3165(86)90019-1.

15 Graham Cormode, Minos N. Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends Databases*, 4(1-3):1–294, 2012. doi:10.1561/1900000004.

16 Saumya K. Debray and Nai-Wei Lin. Static estimation of query sizes in horn programs. In Serge Abiteboul and Paris C. Kanellakis, editors, *ICDT'90, Third International Conference on Database Theory, Paris, France, December 12-14, 1990, Proceedings*, volume 470 of *Lecture Notes in Computer Science*, pages 514–528. Springer, 1990. doi:10.1007/3-540-53507-1_99.

17 Kyle Deeds, Dan Suciu, Magda Balazinska, and Walter Cai. Degree sequence bound for join cardinality estimation, 2022. arXiv:2201.04166.

18 Alin Dobra. Histograms revisited: when are histograms the best approximation method for aggregates over joins? In Chen Li, editor, *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*, pages 228–237. ACM, 2005. doi:10.1145/1065167.1065196.

19 Cristian Estan and Jeffrey F. Naughton. End-biased samples for join cardinality estimation. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 20. IEEE Computer Society, 2006. doi:10.1109/ICDE.2006.61.

20 Ehud Friedgut and Jeff Kahn. On the number of copies of one hypergraph in another. *Israel J. Math.*, 105:251–256, 1998. doi:10.1007/BF02780332.

21 Allen Van Gelder. Multiple join size estimation by virtual domains. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC, USA*, pages 180–189. ACM Press, 1993. doi:10.1145/153850.153872.

22 Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16, 2012. doi:10.1145/2220357.2220363.

23 Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. Fixed-precision estimation of join selectivity. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC, USA*, pages 190–201. ACM Press, 1993. doi:10.1145/153850.153875.

24 Peter J. Haas, Jeffrey F. Naughton, and Arun N. Swami. On the relative cost of sampling for join selectivity estimation. In Victor Vianu, editor, *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 24-26, 1994, Minneapolis, Minnesota, USA*, pages 14–24. ACM Press, 1994. doi:10.1145/182591.182594.

25 Hazar Harmouch and Felix Naumann. Cardinality estimation: An experimental survey. *Proc. VLDB Endow.*, 11(4):499–512, 2017. doi:10.1145/3186728.3164145.

26 Axel Hertzschuch, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. Simplicity done right for join ordering. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org, 2021. URL: http://cidrdb.org/cidr2021/papers/cidr2021_paper01.pdf.

27 Dawei Huang, Dong Young Yoon, Seth Pettie, and Barzan Mozafari. Join on samples: A theoretical guide for practitioners. *Proc. VLDB Endow.*, 13(4):547–560, 2019. doi:10.14778/3372716.3372726.

28 Sungjin Im, Ben Moseley, Hung Q. Ngo, Kirk Pruhs, and Alireza Samadian. On the complexity of computing the polymatroid bound, 2022. Manuscript.

29 Yannis E. Ioannidis. The history of histograms (abridged). In Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer, editors, *Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, September 9-12, 2003*, pages 19–30. Morgan Kaufmann, 2003. doi:10.1016/B978-012722442-8/50011-2.

30 Yannis E. Ioannidis and Stavros Christodoulakis. On the propagation of errors in the size of join results. In James Clifford and Roger King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 29-31, 1991.*, pages 268–277. ACM Press, 1991. doi:10.1145/115790.115835.

31 H. V. Jagadish, Raymond T. Ng, and Divesh Srivastava. Substring selectivity estimation. In Victor Vianu and Christos H. Papadimitriou, editors, *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*, pages 249–260. ACM Press, 1999. doi:10.1145/303976.304001.

32 Sai Vikneshwar Mani Jayaraman, Corey Ropell, and Atri Rudra. Worst-case optimal binary join algorithms under general ℓ_p constraints. *CoRR*, abs/2112.01003, 2021. arXiv:2112.01003.

33 E. T. Jaynes. Information theory and statistical mechanics. *Phys. Rev. (2)*, 106:620–630, 1957.

34 Stasys Jukna. *Extremal combinatorics*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg, second edition, 2011. With applications in computer science. doi:10.1007/978-3-642-17364-6.

35 Hossein Keshavarz, Hung Q. Ngo, and XuanLong Nguyen. Output size bounds under histogrammed frequency moment constraints, 2022. Manuscript.

36 Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org, 2019. URL: http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf.

37 Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Optimal histograms for hierarchical range queries. In Victor Vianu and Georg Gottlob, editors, *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, pages 196–204. ACM, 2000. doi:10.1145/335168.335223.

38 Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, 2015. doi:10.14778/2850583.2850594.

39 Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. Cardinality estimation done right: Index-based join sampling. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org, 2017. URL: http://cidrdb.org/cidr2017/papers/p9-leis-cidr17.pdf.

40 Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *VLDB J.*, 27(5):643–668, 2018. doi:10.1007/s00778-017-0480-7.

41 Richard J. Lipton and Jeffrey F. Naughton. Query size estimation by adaptive sampling. In Daniel J. Rosenkrantz and Yehoshua Sagiv, editors, *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*, pages 40–46. ACM Press, 1990. doi:10.1145/298514.298540.

42 G. Lohman. Is query optimization a solved problem?, 2014. URL: http://wp.sigmod.org/?p=1075.

43 Volker Markl, Peter J. Haas, Marcel Kutsch, Nimrod Megiddo, Utkarsh Srivastava, and Tam Minh Tran. Consistent selectivity estimation via maximum entropy. *VLDB J.*, 16(1):55–76, 2007. doi:10.1007/s00778-006-0030-1.

44 Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proc. VLDB Endow.*, 2(1):982–993, 2009. doi:10.14778/1687627.1687738.

45 M. Muralikrishna and David J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In Haran Boral and Per-Åke Larson, editors, *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 1-3, 1988*, pages 28–36. ACM Press, 1988. doi:10.1145/50202.50205.

46 S. Muthukrishnan, Viswanath Poosala, and Torsten Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 236–256. Springer, 1999. doi:10.1007/3-540-49257-7_16.

47 Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. 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 111–124. ACM, 2018. doi:10.1145/3196959.3196990.

48 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 37–48. ACM, 2012.

49 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *PODS*, pages 37–48, 2012.

50 J. Radhakrishnan. Entropy and counting. In J. C. Misra, editor, *Computational Mathematics, Modelling and Algorithms*, pages 146–168. Narosa Pub House, 2003.

51 Raghu Ramakrishnan and Johannes Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.

52 Christopher Ré and Dan Suciu. Understanding cardinality estimation using entropy maximization. In Jan Paredaens and Dirk Van Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 53–64. ACM, 2010. doi:10.1145/1807085.1807095.

53 P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD '79, pages 23–34, New York, NY, USA, 1979. ACM.

54 Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In Philip A. Bernstein, editor, *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, USA, May 30 - June 1*, pages 23–34. ACM, 1979. doi:10.1145/582095.582099.

55 Richard P. Stanley. *Enumerative combinatorics. Volume 1*, volume 49 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, second edition, 2012.

56 Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 96–106. OpenProceedings.org, 2014. doi:10.5441/002/icdt.2014.13.

57 Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. Deep unsupervised cardinality estimation. *Proc. VLDB Endow.*, 13(3):279–292, 2019. doi:10.14778/3368289.3368294.

58 Raymond W. Yeung. *Information Theory and Network Coding*. Springer Publishing Company, Incorporated, 1 edition, 2008.

59 Zhen Zhang and Raymond W. Yeung. On characterization of entropy function via information inequalities. *IEEE Transactions on Information Theory*, 44(4):1440–1452, 1998. doi:10.1109/18.681320.

Counting the Solutions to a Query

Marcelo Arenas ✉
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute Foundational Research on Data, Santiago, Chile

Abstract

In this talk, we consider the problem of counting the solutions to a query. Our first motivating scenario is the use of regular expressions to extract paths from a graph database. More specifically, given a graph database D, a regular expression r and a natural number n, consider the problem of counting the number of paths p in D such that p conforms to r and the length of p is n. This problem is known to be hard, namely #P-complete. In this talk, we show that this problem admits a fully polynomial-time randomized approximation scheme (FPRAS). Remarkably, the key idea to prove this result is to show that the fundamental problem #NFA admits an FPRAS, where #NFA is the problem of counting the number of strings of length n accepted by a non-deterministic finite automaton (NFA). While this problem is known to be #P-complete and, more precisely, SpanL-complete, it was open whether this problem admits an FPRAS. In this work, we solve this open problem and obtain as a welcome corollary that every function in SpanL admits an FPRAS.

As a second motivating scenario, we consider the widely used class of conjunctive queries over relational databases. More specifically, for every class $\mathcal{C}$ of conjunctive queries with bounded treewidth, we introduce the first FPRAS for counting the answers to a query in $\mathcal{C}$. In fact, our FPRAS is more general, and also applies to conjunctive queries with bounded hypertree width, as well as unions of such queries. As for the case of graph databases, the key ingredient in our proof is the resolution of a fundamental counting problem from automata theory. Specifically, we show that the problem #TA admits an FPRAS, where #TA is the problem of counting the number of trees of size n accepted by a tree automaton (TA).

This talk is based on the results presented in [1, 2].

2012 ACM Subject Classification Information systems → Graph-based database models; Information systems → Query languages; Theory of computation → Regular languages; Theory of computation → Tree languages

Keywords and phrases Counting, query answering, fully polynomial-time randomized approximation scheme

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.2

Category Invited Talk

Funding *Marcelo Arenas*: This work was funded by ANID–Millennium Science Initiative Program–Code ICN17_002, and by Fondecyt grant 1191337.

References

1 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. #NFA admits an FPRAS: efficient enumeration, counting, and uniform generation for logspace classes. *J. ACM*, 68(6):48:1–48:40, 2021.

2 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. When is approximate counting for conjunctive queries tractable? In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1015–1027, 2021.

Answering Unions of Conjunctive Queries with Ideal Time Guarantees

Nofar Carmeli
DI ENS, ENS, CNRS, PSL University, Paris, France
Inria, Paris, France

Abstract

The holy grail we strive for is, given a query, to identify an algorithm that answers it over general databases with optimal time guarantees for the specific query. In this tutorial, we focus on what can be seen as ideal time guarantees: linear preprocessing (needed to read the input) and constant time per answer (needed to print the output). We seek to understand which queries can be solved with these (or almost these) time guarantees and how.

We start with the basic building blocks of database queries: joins, and slowly increase the expressivity by introducing projections and unions until we cover positive relational algebra. We first consider the task of enumerating all query answers and then discuss related, more demanding, tasks such as ordered enumeration and direct access to query answers. We investigate the challenges in answering such queries and provide algorithms and conditional lower bounds

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

Keywords and phrases query evaluation, enumeration, fine-grained complexity, constant delay, union of conjunctive queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.3

Category Invited Talk

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier; Article No. 3; pp. 3:1–3:1

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

On the Hardness of Category Tree Construction

Shay Gershtein
Tel Aviv University, Israel

Uri Avron
Tel Aviv University, Israel

Ido Guy
Ben-Gurion University of the Negev, Beer Sheva, Israel

Tova Milo
Tel Aviv University, Israel

Slava Novgorodov
eBay Research, Netanya, Israel

Abstract

Category trees, or taxonomies, are rooted trees where each node, called a category, corresponds to a set of related items. The construction of taxonomies has been studied in various domains, including e-commerce, document management, and question answering. Multiple algorithms for automating construction have been proposed, employing a variety of clustering approaches and crowdsourcing. However, no formal model to capture such categorization problems has been devised, and their complexity has not been studied. To address this, we propose in this work a combinatorial model that captures many practical settings and show that the aforementioned empirical approach has been warranted, as we prove strong inapproximability bounds for various problem variants and special cases when the goal is to produce a categorization of the maximum utility.

In our model, the input is a set of n weighted item sets that the tree would ideally contain as categories. Each category, rather than perfectly match the corresponding input set, is allowed to exceed a given threshold for a given similarity function. The goal is to produce a tree that maximizes the total weight of the sets for which it contains a matching category. A key parameter is an upper bound on the number of categories an item may belong to, which produces the hardness of the problem, as initially each item may be contained in an arbitrary number of input sets.

For this model, we prove inapproximability bounds, of order $\tilde{\Theta}(\sqrt{n})$ or $\tilde{\Theta}(n)$, for various problem variants and special cases, loosely justifying the aforementioned heuristic approach. Our work includes reductions based on parameterized randomized constructions that highlight how various problem parameters and properties of the input may affect the hardness. Moreover, for the special case where the category must be identical to the corresponding input set, we devise an algorithm whose approximation guarantee depends solely on a more granular parameter, allowing improved worst-case guarantees. Finally, we also generalize our results to DAG-based and non-hierarchical categorization.

2012 ACM Subject Classification Theory of computation → Data structures and algorithms for data management; Theory of computation → Approximation algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases maximum independent set, approximation algorithms, approximation hardness bounds, taxonomy construction, category tree construction

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.4

Related Version *Full Version*: https://slavanov.com/research/icdt22_full.pdf

1 Introduction

Category trees, or taxonomies, are rooted trees where each node corresponds to a labeled category defined as a set of related items. Each non-leaf category is more general than its descendants and contains the union of their item sets. Moreover, each item may typically appear in a bounded number of tree branches. Such trees enable browsing-style information access and play a central role in Web platforms. While taxonomists can identify many desirable categories, producing a single categorization in a compact structure to maximize a given utility measure is challenging. Therefore, multiple algorithms for automating construction in various domains, e.g., e-commerce [2, 12], document management [9], and question answering [24], have been proposed, employing a variety of clustering approaches, and crowdsourcing [9, 21]. However, to our knowledge, the complexity of this problem has not been studied w.r.t. a formal model, and solution evaluations were based on user-studies or a similarity score of the tree categories to a collection of desired categories [17, 9, 18], to measure how well these are captured by the much more succinct solution. Based on the latter evaluation method, we propose a model that captures practical settings and show that the aforementioned heuristic approach has been warranted, as we prove strong inapproximability bounds.

Before describing our results, we first define the formal setting.

Model. The input is a set of n sets of items. The solution space consists of rooted trees (we also examine other structures, as described in the sequel). Each node (category) corresponds to a set of items (not necessarily identical to any set in the input), and every non-leaf category contains the union of all the items of its descendants. Ideally, the tree would have, for every input set, a category that is very similar to it. Each input set is weighted to reflect how valuable it is for a solution to contain a matching category. In practice, an input set represents items that match some criteria a user may have in mind when performing a search, and its weight implies the predicted likelihood of seeking these criteria. The sets are derived from a dataset of result sets to search queries, or, more generally, are formed by grouping items w.r.t. shared properties.

This model has multiple variants, defined by two parameters. The first parameter is a *similarity function*, which measures the similarity of an input set and a category. We examine several variations of commonly used set-similarity functions, which extend the original function with a threshold parameter. A similarity score below this parameter is rounded down to 0, to capture the fact that, when the similarity score is too low, no category is identifiable by the user as a matching category, and has no utility. Given such a function, the tree score for a given input set is the maximum similarity score of any category for this set. The overall tree score is the weighted (w.r.t. input weights) sum of the scores for all the input sets. The goal is to produce a tree of the highest score.

The second parameter is a *copy-bound*, which limits the number of *independent* categories any item can belong to, where categories are called independent if no two are on the same path from the root to a leaf. Most real-life platforms set a low copy-bound, to ensure that the categorization is coherent, compact, and easy to navigate. For example, eBay allows listing an item in a single (lowermost) category for free, or two categories for an extra fee [1].

Our bounds also apply to the related problems, where the aim is to produce a flat categorization or more general DAG structures. Flat categorization may be of independent interest, as it also captures the setting where one seeks, given a collection of overlapping sets, a partition that is maximally similar to the original collection. This may be particularly relevant for clustering and partitioning problems in hypergraphs (see Section 7).

Results. For the optimization problem of maximizing the tree score (as defined above), we prove for all examined variants an inapproximability bound of $\tilde{\Theta}(\sqrt{n})$ or $\tilde{\Theta}(n)$, where n is the number of input sets, highlighting how different problem parameters may affect the hardness. These bounds also apply to unweighted inputs and various special cases. On the other hand, we show that finer properties, such as bounds on the cardinality of input sets or the number of intersections among sets, aid in deriving more relaxed parameterized hardness bounds. To that end, we also provide a positive result in the form of an algorithm for the *Exact variant*, where a category must match an input set exactly to contribute to the objective function. The tight performance guarantee of this algorithm depends only on a parameter that measures the number of intersections between the input sets. Importantly, we reduce this variant to the Maximum Independent Set problem, for which, despite its inapproximability, practical solvers have been devised. We demonstrate the practical utility of this result, in [3] and [4], complementary works focusing on constructing an e-commerce category tree that is maximally similar to result sets of user queries. In these works, we show that leveraging these solvers enables finding optimal solutions to real-world instances and extend this approach to algorithms that solve well instances of more general variants.

An essential component in our methods is defining a generalized similarity function with two threshold parameters that address two more granular similarity measures: *precision* and *recall*. Our key results consist of reductions from the Maximum Independent Set problem in hypergraphs, where we integrate into the reduced instance randomized constructions that closely capture the precision and recall parameters. This not only enables us to derive improved results for more subtle special cases but also to capture more refined properties of hard inputs, which we then leverage to prove hardness w.r.t. other similarity functions (we outline how to schematically apply our arguments to derive hardness bounds for similarity functions not examined here).

While we are not aware of any theoretical results directly comparable to ours, Section 7 discusses motivating empirical research and possible applications to hypergraph partitioning.

Lastly, we note that the complementary problem of labeling the resulting categories has been studied in various settings (e.g., [5]), and is outside the scope of our model.

Outline. Section 2 provides the necessary formalism for our model, while Section 3 presents useful theoretical tools. In Section 4 we provide a positive result for a common problem variant. In Section 5 we prove various approximation hardness results derived w.r.t. the generalized similarity function (with recall and precision thresholds). In Section 6 we leverage these results to derive hardness bounds w.r.t. all other similarity functions defined in Section 2. The related work appears in Section 7 and we conclude in Section 8.

Due to space constraints, we defer all formal proofs to the full version of the paper [10].

2 Model

We now define the model underlying our work, followed by a discussion of problem parameters. We conclude this section with illustrative examples of problem instances in our model.

2.1 Problem definition

The two problems we study are the Optimal Category Tree problem (OCT) and the Optimal Category Partition problem (OCP). The input to both problems is $\langle Q, U, W \rangle$, where $Q \subseteq 2^U$ is a set of n sets over a finite universe U, and $W : Q \to [0, 1]$ is a weighting function that assigns a non-negative weight to each set in Q. We use the term *query*, to denote each set

in Q. Note, in advance, that in the definition of the model we discuss two types of element sets: the queries and the sets corresponding to the tree nodes. In general, these sets are not identical (however, these are typically similar, as the objective is, roughly speaking, to maximize the similarity of the two types of sets, as defined formally below).

Both problems have multiple variants defined by two parameters (explained below, in the context of the solution space): a *copy bound* $r \in \mathbb{N}$, and a *similarity function* $\mathcal{S} : [2^U] \times [2^U] \to [0, 1]$. We denote by $OCT^r(\mathcal{S})$ the r-copy OCT problem with similarity function $\mathcal{S}$, and the analogous OCP variant is denoted by $OCP^r(\mathcal{S})$.

We next formally define the solution space for each of the two problems. We start with $OCP^r(\mathcal{S})$, as it is a simpler form of the model for $OCT^r(\mathcal{S})$.

OCP. We call a set of sets over U an *r-weak partition*, if every element appears in at most r sets. A 1-weak partition is a standard partition. The solution space of $OCP^r(\mathcal{S})$ consists of all r-weak partitions of U. Any such solution is termed as a *category partition*, denoted by P, with the sets contained in it termed as *categories*.

Given a query, $q \in Q$, and a category partition, P, we define the *similarity score* of a category $C \in P$ for q as $\mathcal{S}(q, C)$. The score of P for this query is defined by the category that most closely matches the query as $S(q, P) = \max_{C \in P} \mathcal{S}(q, C)$. Note that the root of the tree, as it is a valid category, may also be the most closely matching category for some queries. The overall score of the category partition is defined as $S(Q, P) = \sum_{q \in Q} W(q) \cdot S(q, P)$. This score is the weighted sum of the scores for all queries, where the weight of each score is the corresponding query weight. The objective of the $OCP^r(\mathcal{S})$ problem is to produce a category partition of the maximum score: $\arg\max_P S(Q, P)$.

OCT. The solution space of $OCT^r(\mathcal{S})$ consists of rooted trees, termed *category trees*, where every tree node, termed *category*, contains a subset of U. We abuse notation, and, when clear from context, we use T to denote both the category tree and the set of its categories. Similarly, we use C to denote a category as well as the set of elements it contains.

A category tree must satisfy the following two requirements. First, every non-leaf category contains the union of the sets of elements contained by its child categories (and possibly other elements). The root of the tree, thus, contains all the elements that appear in any category. Second, for each element $e \in U$ there are at most r semi-leaves w.r.t. e, where the semi-leaves w.r.t. e are the most specific categories to which e belongs (i.e. none of the descendants of any such semi-leaf contain e). Note that for a category tree, unlike a category partition, it is no longer true (nor desirable) that an element is contained in at most r categories. Even for $r = 1$, if an element is contained in some category in the tree, it must also be contained in all its ancestor categories. Therefore, the copy-bound is applied to the number of semi-leaves w.r.t. any given element, with the only other nodes containing the element being all the ancestors of these semi-leaves. For $r = 1$ this requirement implies that any given element in the tree is contained only in categories that are all on the same branch.

All definitions of relevant scoring functions are analogous to $OCP^r(\mathcal{S})$. Concretely, the score of a tree, T, for a query, q, is defined as $S(q, T) = \max_{C \in T} \mathcal{S}(q, C)$. The overall score of T is $S(Q, T) = \sum_{q \in Q} W(q) \cdot S(q, T)$. When Q is clear from context, we use the shorthand $S(T)$. The objective of the $OCT^r(\mathcal{S})$ problem is to produce $\arg\max_T S(Q, T)$.

Unweighted variants. We refer to the special case of OCT (OCP) where all weights are uniform as *unweighted* OCT (OCP) and set all weights to 1. Our hardness proofs leverage unweighted inputs, and therefore our hardness bounds also apply to the unweighted case. Accordingly, in our hardness discussions, the reader may assume this context. We directly use weights only in the upper bound we provide in Section 4.

2.2 Similarity functions

We study several similarity functions, that are dependent (in some cases, implicitly) on the following two underlying similarity measures, *precision* $p(q, C) = \frac{|C \cap q|}{|C|}$ and *recall* $r(q, C) = \frac{|C \cap q|}{|q|}$. We distinguish between *cutoff functions* and *threshold functions*. Both have a threshold parameter $\delta \in (0, 1]$ and use an underlying similarity function f. In both cases, the function outputs 0 when $f(q, C) < \delta$. However, when $f(q, C) \geq \delta$ a cutoff function equals $f(q, C)$, whereas a threshold function equals 1. We first focus, however, on the following, more general, threshold function, which sets a separate threshold for each measure.

▶ **Definition 1** (Granular threshold function). *Given parameters $\alpha, \beta \in [0, 1]$, the granular threshold similarity $\mathcal{T}_{\alpha,\beta}$ of a query q and category C is defined as follows: $\mathcal{T}_{\alpha,\beta}(q, C) = 1$ when $p(q, C) \geq \alpha$ and $r(q, C) \geq \beta$, and $\mathcal{T}_{\alpha,\beta}(q, C) = 0$ otherwise.*

We will also study the common similarity functions defined below.

▶ **Definition 2** (Jaccard similarity). *The Jaccard similarity of a category C and a query q is defined as $J(q, C) = \frac{|q \cap C|}{|q \cup C|}$. The threshold Jaccard similarity, with threshold parameter $\delta \in (0, 1]$, is defined as $\hat{J}_\delta(q, C) = 1$ when $J(q, C) \geq \delta$ and $\hat{J}_\delta(q, C) = 0$ otherwise. The cutoff Jaccard similarity, with threshold parameter $\delta \in (0, 1]$, is defined as $\bar{J}_\delta(q, C) = J(q, C)$ when $J(q, C) \geq \delta$ and $\bar{J}_\delta(q, C) = 0$ otherwise.*

▶ **Definition 3** (F_1 score). *The F_1 score of a category C for a query q is defined as the harmonic mean of the precision and the recall: $F_1(q, C) = 2\frac{p(q,C) \cdot r(q,C)}{p(q,C)+r(q,C)}$. The threshold F_1 score, with parameter $\delta \in (0, 1]$, is defined as $\hat{F}_{1(\delta)}(q, C) = 1$ when $F_1(q, C) \geq \delta$ and $\hat{F}_{1(\delta)}(q, C) = 0$ otherwise. The cutoff F_1 score, with threshold $\delta \in (0, 1]$, is defined as $\bar{F}_{1(\delta)}(q, C) = F_1(q, C)$ when $F_1(q, C) \geq \delta$ and $\bar{F}_{1(\delta)}(q, C) = 0$ otherwise.*

Cover terminology. If a category C has the highest score for a query q (if necessary, ties are broken arbitrarily), and that score is not 0, we say that C *covers* q. We call a category that covers at least one query a *covering* category, and a branch containing a covering category is a *covering* branch. A set of categories is *independent* if no two categories are on the same branch (OCP categories are independent). Similarly, a set of queries are *independently-covered*, if each is covered by a different category, and the covering categories are independent. Observe that in unweighted instances (where, as noted earlier, all weights are assumed to be 1) with threshold functions the score equals the number of covered queries.

Note that, all functions defined above share the special case of $\mathcal{T}_{1,1}$ (Definition 1) (equivalent to setting $\delta = 1$ in Definitions 2 and 3), where a query q is covered by a category C only if $q = C$. We refer to this variant as the *Exact variant*.

Canonical form. Any category tree can be reduced to a canonical form, without decreasing the score, by (1) removing non-covering categories, (2) connecting the parent and children of any removed category, and (3) removing from category C and its descendants any element not contained in any query covered by C or categories below C (this may even improve the precision and the score). If a query is covered by multiple categories, one can assign arbitrarily a single category that is said to cover it, and then reduce it to a canonical form, as described above, w.r.t. this assignment. Similarly, adding new categories that do not affect the contents of the existing categories cannot decrease the tree score. This discussion applies analogously to category partitions.

Choices of parameters. For practical applicability, we focus on variants where $r = \Theta(1)$ and the similarity functions have threshold parameters. A low copy-bound ensures a concise categorization, and in many platforms, r is a small constant, typically, 1 (e.g., [1]). This parameter controls the trade-off between the score and the conciseness, and our parameterized bounds hint at a quantification of this trade-off. Threshold parameters capture the fact that, below a certain similarity score, a category has no utility. Without thresholds, trees that cover unacceptably poorly all queries may be mathematically preferable to trees that cover well a smaller number of queries. Nevertheless, to capture more tolerant settings, we also provide approximation bounds for polynomially small threshold parameters.

We also note that, in practice, errors in precision and recall have an asymmetric effect. For example, perfect recall with precision of $\frac{1}{2}$, enables the user to examine all relevant items to identify the best matches, while ignoring every other item. This may be acceptable, especially for smaller categories. However, in the analogous case of perfect precision and recall of $\frac{1}{2}$, other categories may or may not contain better matching items, and the user might waste time looking for non-existing or hard-to-find categories or be unaware of better options. It may, therefore, be tempting, in some applications, to require perfect recall. To that end, we examine this case separately and show that it admits the strictest inapproximability.

More generally, there exists a key tension between the recall and precision thresholds. Consider, as an extreme example, a recall threshold of 1, and a precision threshold of 0 (i.e., perfect recall with no precision requirement). For this variant, a tree consisting only of a root that contains all the elements is an optimal solution. At the other extreme, if we require perfect precision with no constraint on the recall, then an optimal solution is a tree where there is a leaf for each element, containing only that element, and a root connected directly to all the leaves. These edge cases illustrate the following intuitive phenomenon: increasing precision thresholds leads to more granular trees with smaller covering categories, whereas increasing recall thresholds generally produces a more coarse categorization with larger covering categories. These properties of the precision and recall thresholds are formalized and leveraged in our hardness proofs.

2.3 Examples of Problem Instances

We illustrate the OCT setting with $r = 1$ via the following toy examples, depicted in Figure 1. The figure presents two optimal solutions, computed by brute-force, corresponding to two different OCT variants, over the same input, provided on the left side. For convenience, as it is easier to perform arithmetic with integers, we provide integer weights, instead of normalizing into $[0, 1]$, since the normalization of the weights and scores does not affect the complexity of the problem or the performance ratio of the various solutions.

Observe that the overall weight of all four queries is 5, hence this is also an upper bound on the score of any tree for any variant over this input. In addition, observe that, since $r = 1$, we cannot add any branches to either of the depicted trees, without violating the copy-bound constraint.

▶ **Example 4.** The tree T_1, depicted in the middle of the figure, is the optimal solution for the $OCT^1(\mathcal{T}_{0.8,1})$ variant (i.e. precision 0.8 and perfect recall). The categories C_3 and C_4 cover the queries q_2 and q_3, respectively, as they are identical to these queries (and would cover them even for $\alpha = 1$). The category C_1 covers q_1 as its recall score is 1, and 5 out of the 6 items in C_1 are in q_1, hence the precision is $\frac{5}{6} > \alpha$. Note that, we must include f in C_1 since it appears in C_4, and removing f from both categories, would result in C_4 no longer covering q_3. Moreover, there is no incentive to place f elsewhere, since the score, when using a binary function, is not penalized for precision errors if the threshold is exceeded.

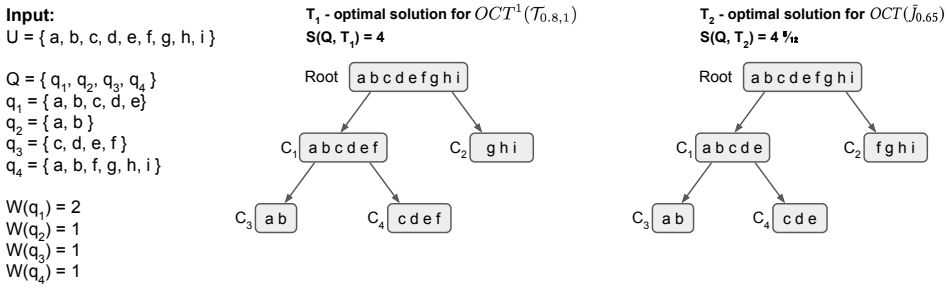


Figure 1 Optimal solutions for two OCT variants over the same input (where, for simplicity, the input weights are not normalized), depicted on the left side. The category tree, T_1, is an optimal solution for the $OCT^1(\mathcal{T}_{0.8,1})$ variant, where C_1 covers q_1, C_3 covers q_2, and C_4 covers q_3, with the overall score of $W(q_1) + W(q_2) + W(q_3) = 4$. The rightmost tree, T_2, is the optimal solution for the cutoff Jaccard variant with $\delta = 0.6$, where C_1 covers q_1 with the score of 1, C_2 covers q_4 with the score of $\frac{2}{3}$, C_3 covers q_2 with the score of 1, and C_4 covers q_3 with the score of $\frac{3}{4}$, resulting in the overall score of $W(q_1) \cdot 1 + W(q_2) \cdot 1 + W(q_3) \cdot \frac{3}{4} + W(q_4) \cdot \frac{2}{3} = 4\frac{5}{12}$.

As for the category C_2, its addition to the tree is optional, since it does not cover any query, despite all its items belonging to the uncovered query, q_4, as we can no longer achieve perfect recall without the items $\{a, b, f\}$. It is easy to verify that there is no way to cover q_4 by adding a matching category above or below C_1, such that the items $\{a, b, f\}$ would be shared by all categories, without decreasing the precision of other queries to values below the threshold.

▶ **Example 5.** We next discuss, T_2, the optimal solution for $OCT^1(\bar{J}_{0.65})$, the cutoff Jaccard variant with $\delta = 0.65$, depicted on the right side of Figure 1. It overlaps with T_1, except for the item f, which is placed in C_2 instead of C_4 and C_1. In this case, compared to the previously examined variant, since Jaccard variants allow for errors in both precision and recall, and also since we use a lower threshold, it is now possible to cover all queries, albeit with imperfect scores. Indeed, every non-root category in T_2 covers a query, as explained in the figure. Moreover, q_1 is the query of the maximal weight, hence it is not surprising that the optimal tree covers it with a perfect score, at the expense of errors in the covers of less significant queries. We note that, in practice, the same category often covers multiple queries. For instance, if we decrease the threshold from 0.65 to 0.4, then C_1 would also cover q_2, as its precision w.r.t. q_2 is exactly 0.4.

3 Preliminaries

We provide here known results and definitions, that we will use in our hardness proofs. We conclude the section by explaining how proofs are tailored to fit both OCT and OCP simultaneously, and discuss generalizing a tree to a DAG.

Notation. To simplify the presentation, we use a "soft-omega" notation, $\tilde{\Theta}(\cdot)$, to hide sub-polynomial factors. Whenever we state that a variant has inapproximability of $\tilde{\Theta}(n^c)$, for some constant $c \in (0, 1]$, this compact notation implies the more formal argument that, for any $\epsilon > 0$, this variant cannot be approximated within a factor of $O(n^{c-\epsilon})$. We note that a solution of score 1 can always be achieved by producing a single category that equals one of the queries (for differently weighted queries we will specifically select the query of the highest weight). Thus, $\tilde{\Theta}(n)$ is the strictest possible inapproximability factor, using this notation.

Complexity Assumptions. We next define the complexity class ZPP, as some of our results use the assumption $ZPP \neq NP$. It is known that $P \subseteq ZPP \subseteq NP$ and that $ZPP \subseteq BPP$, where BPP is the class of problems solvable by a randomized PTIME algorithm with a two-sided error.

▶ **Definition 6.** *The complexity class ZPP contains the problems for which there is a PTIME algorithm that outputs DO NOT KNOW with a probability of less than $1/2$, and outputs the correct answer with the remaining probability.*

MIS. We leverage reductions from the Maximum Independent Set problem (MIS) in *uniform hypergraphs*. In an r-uniform hypergraph, all (hyper)edges are vertex subsets of cardinality r. The special case of $r = 2$ is a graph.

▶ **Definition 7.** *In the Maximum Independent Set problem (MIS) in uniform hypergraphs, the input is a uniform hypergraph $G = (V, E)$, and the objective is to find a vertex set $S \subseteq V$ of maximum cardinality, subject to the constraint that no edge from E is contained in S.*

We have made use of the following known results for MIS, where $n = |V|$.

▶ **Theorem 8** ([11])**.** *The MIS problem in r-uniform hypergraphs, for constant $r \geq 2$, cannot be approximated below a $\tilde{\Theta}(n)$ factor, unless $ZPP = NP$.*

▶ **Theorem 9** ([27], [6])**.** *The MIS problem in graphs has inapproximability of $\tilde{\Theta}(n)$, unless $P = NP$. Moreover, for graphs of sufficiently large constant degree bound d, MIS is hard to approximate below a $\Theta(\frac{d}{\log^2 d})$ factor, unless $BPP = NP$. Furthermore, MIS is NP-hard even for regular graphs of degree 3.*

▶ **Theorem 10** ([7])**.** *For r-uniform hypergraphs with (not necessarily constant) maximum degree d, there exists a $PTIME$ algorithm producing an independent set of size $\Omega(\frac{n}{d^{\frac{1}{r-1}}})$.*

From Theorem 10, we derive the following lemma.

▶ **Lemma 11.** *Given an r-uniform hypergraph $G = (V, E)$, there exists a PTIME algorithm that produces an independent set in G of size $\Omega((\frac{|V|^r}{|E|})^{\frac{1}{r-1}})$.*

Proof of Lemma 11. The average degree of G is $\bar{d} = \frac{r|E|}{|V|}$. Let $V_1 \subseteq V$ denote the set of vertices in G whose degree is at most $d = 2\bar{d}$. A simple counting argument implies that $|V_1| \geq \frac{|V|}{2}$. Consider the sub-hypergraph G_1 of G induced by V_1. Computing G_1 is the first step of the algorithm.

In the second and last step, we apply over G_1 the algorithm from Theorem 10, which produces an independent set S. By Theorem 10, the size of this independent set is

$$|S| \geq \Omega(\frac{\frac{|V|}{2}}{d^{\frac{1}{r-1}}}) = \Omega(\frac{|V|}{(\frac{r|E|}{|V|})^{\frac{1}{r-1}}}) = \Omega((\frac{|V|^r}{|E|})^{\frac{1}{r-1}}).$$ ◀

Hard instances of MIS. When reducing from MIS, we will restrict ourselves to instances where the optimal solution is of size $\tilde{\Theta}(n)$. The $\tilde{\Theta}(n)$ inapproximability of MIS implies that this subset of inputs captures the maximal hardness. Accordingly, in our reductions, assuming this hard set of inputs, we will leverage the fact that one cannot find (in the worst case) an independent set of size $\Omega(n^\epsilon)$ for any $\epsilon > 0$.

Probabilistic Tools. We next define the Hypergeometric and Binomial distributions and present known tail bounds for both. These are useful in the analysis of our randomized reduction.

▶ **Definition 12.** *Consider sampling without replacement n uniformly random and independent samples from a set of N elements containing K special elements, and let X denote the number of special elements in the sample. Then, X is a hypergeometric random variable, denoted as $X \sim H(N, K, n)$, and its probability mass function is* $\Pr(X = i) = \frac{\binom{K}{i}\binom{N-K}{n-i}}{\binom{N}{n}}$.

▶ **Definition 13.** *Consider performing n independent experiments with success probability p. Let X denote the number of successful experiments. Then, X is a binomial random variable, denoted as $X \sim B(n, p)$, with probability mass function is* $\Pr(X = i) = \binom{n}{i} p^i (1-p)^{n-i}$.

We use the following tail-bound for the hypergeometric distribution.

▶ **Lemma 14** ([20]). *If $X \sim H(N, K, n)$, as defined in Definition 12, and letting $u = \frac{K}{N}$, then, for $t > 0$:*

$$\Pr(X \geq (u+t)n) \leq \left(\left(\frac{u}{u+t}\right)^{u+t}\left(\frac{1-u}{1-u-t}\right)^{1-u-t}\right)^{n}.$$

We also use the following Chernoff bound for the binomial distribution.

▶ **Lemma 15** (Chernoff Bound [23]). *If $X \sim B(n, p)$, as defined in Definition 13, then, denoting the expectation $\mu = np$, for $\delta \geq 1$:*

$$\Pr(X > (1+\delta)\mu) < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}.$$

OCP and DAGs. Finally, we explain how our hardness reductions for OCT were devised to also apply for OCP, as well as for the more general problem where one is allowed to produce a rooted DAG with analogous combinatorial constraints. Thus, while the hardness analysis focuses on OCT, *the bounds apply for the above two problems as well.*

Loosely speaking, an algorithm that produces a tree has all the capabilities it would need for producing a similar-score partition, along with several additional possibilities to increase the score. Hence, in our analysis, by bounding what is possible for constructing a tree, we also bound what is possible for constructing a partition.

Concretely, observe that, over any given input, any category partition can be transformed into a category tree of the same score, by connecting all the categories to a root. In particular, over any given input, the optimal score, that can be achieved by a category partition, cannot exceed the optimal score by a category tree. Importantly, in all examined variants, we ensure that our hardness bounds are derived over a subset of inputs for which there exists a category tree whose leaf categories induce a category partition of score $\tilde{\Theta}(n)$, implying that the optimal score of both problems is of at least this order, which is roughly maximal (the score for any input cannot exceed $n = |Q|$). It follows that all our approximation hardness bounds for OCT hold for OCP as well.

We note that our hardness proofs also apply to the more general problem where instead of a tree, one is allowed to produce any rooted DAG, maintaining the requirements that a category must contain all its descendants and for each element e there are at most r different paths from the root to a semi-leaf w.r.t. e (recall that a semi-leaf w.r.t. e is a most specific node to which e belongs). This follows from the fact the any such DAG can be converted to a valid tree solution, by removing edges, which does not affect the score.

4 Approximation Algorithm for the Exact Variant

Before diving into the hardness analyses, we provide a positive result based on PTIME approximation algorithms for the Exact variant of (weighted) $OCT^1(\mathcal{T}_{1,1})$ and $OCP^r(\mathcal{T}_{1,1})$. Note that for OCT the algorithm applies when $r = 1$, while for OCP it applies to any constant r. The Exact variant is of special interest because it is a special case of all variants pertaining to all examined similarity functions, where the error threshold is $\delta = 1$ (or $\alpha = \beta = 1$ for $\mathcal{T}_{\alpha,\beta}$). We note that in [3] we show empirically that this algorithm can solve real-world instances optimally, using as a subroutine modern weighted MIS solvers, and extend it to algorithms suited for the more general OCT variants. Moreover, as mentioned, the requirement $r = 1$ is the most prevalent in practice.

In Theorem 17 we prove that it is NP-hard to approximate this variant below a $\tilde{\Theta}(n)$ factor. Hence, one cannot provide any non-trivial approximation guarantees for the general case. Nevertheless, we devise an algorithm with an optimal approximation guarantee of $\tilde{O}(\bar{D})$, where $\bar{D} \in [0, n]$, referred to as the *average (weighted) degree* of the input, is a parameter relating to the number of intersections among the queries. Formally, we define a *conflict* as any pair of queries that intersect and neither is a subset of the other (for OCP only the former condition is relevant). The *degree* $d(q)$ of a query $q \in Q$ is defined as the number of conflicts in the input that contain q. The average degree is the weighted average of all query degrees in the input. That is, $\bar{D} = \frac{\sum_{q \in Q} W(q) \cdot d(q)}{\sum_{q \in Q} W(q)}$.

The proof of the following theorem and the description of the algorithms, which are based on the connection to the weighted MIS problem, appear in the full version of the paper [10].

▶ **Theorem 16.** *There exist $\tilde{\Theta}(\bar{D})$-approximation algorithms for $OCT^1(\mathcal{T}_{1,1})$ and $OCP^r(\mathcal{T}_{1,1})$. This factor is optimal (up to negligible factors), unless $P = NP$.*

The result above shows that the approximation hardness is strongly dependent on the average number of intersections between queries, and indeed, in the subsequent hardness analysis, the only practical bound corresponds to a special case where $\tilde{O}(\bar{D})$ is low (Theorem 17).

We remark that in all examined eBay datasets the average degree did not exceed $\log n$, even for high values of the maximum degree. However, we leave an in-depth empirical evaluation along with devising algorithms for other problem variants to future work.

5 Hardness of $OCT^r(\mathcal{T}_{\alpha,\beta})$

In this section, we prove approximation hardness bounds on $OCT^r(\mathcal{T}_{\alpha,\beta})$ for various ranges of the threshold parameters. We first provide a reduction from MIS to $OCT^r(\mathcal{T}_{\alpha,\beta})$ where $\alpha = \Theta(1)$ and $\beta > \frac{1}{2}$, proving $\tilde{\Theta}(n^{\frac{1}{r+1}})$ inapproximability (n is the number of queries, and r is the copy-bound), unless $ZPP = NP$. For the special case of $OCT^r(\mathcal{T}_{\alpha,1})$ we improve this bound to $\tilde{\Theta}(n)$. For $r = 1$, we strengthen these bounds by using a weaker theoretical assumption and also provide a bound for the case of queries of bounded size.

To prove that the $\tilde{\Theta}(n^{\frac{1}{r+1}})$ inapproximability extends to the case where $\beta \leq \frac{1}{2}$, we use a more involved randomized reduction, and also provide an analysis that captures sub-constant ranges of the threshold parameters to derive a $\tilde{\Theta}((\alpha^{(r+2)}\beta n)^{\frac{1}{r+1}})$ inapproximability bound.

The remainder of this section consists of two subsections, pertaining to the two reductions. Each subsection is further divided into the reduction from MIS, the hardness results it implies, and the intuition underlying the proof. We also explain why different reductions were necessary. All formal proofs are appear in the full version of the paper [10].

5.1 Special cases with $\beta > \frac{1}{2}$

We now describe and analyze the first reduction.

Reduction from MIS. Given an algorithm for $OCT^r(\mathcal{T}_{\alpha,\beta})$, denoted by A, with a (worst-case) approximation guarantee of γ, we devise an algorithm $R = R_A$ for MIS in $(r+1)$-uniform hypergraphs. We compute a lower bound on the size of the independent set (IS) that R produces as a function of the approximation guarantee γ. This implies a lower bound on γ, below which R would produce an IS of size $\Theta(poly(n))$ (n is the number of vertices in the hypergraph), contradicting the hardness of MIS.

The algorithm R consists of a sequence of three procedures, R_1, R_2, and R_3:

1. Given an $(r+1)$-uniform hypergraph, $G = (V, E)$, R_1 transforms it into an instance Q of $OCT^r(\mathcal{T}_{\alpha,\beta})$. The universe of elements for Q consists of three types of elements: an *edge element* for every edge in E, *padding elements*, and $\frac{1-\beta}{2\beta-1}n^r$ *joint elements*. Specifically, for each vertex $v \in V$, we construct a query, q_v, such that $Q = \{q_v \mid v \in V\}$. Every query contains all the joint elements. Moreover, each query, q_v, also contains all the edge elements that correspond to edges incident to v in G. Finally, we add to every query as many unique padding elements as necessary, such that the size of the query is $\frac{n^r}{2\beta-1}$. Every padding element appears in only one query. It follows that every query contains $\frac{1-\beta}{2\beta-1}n^r$ joint elements and $\frac{\beta}{2\beta-1}n^r$ non-joint elements.
2. R_2 consists simply of running A over Q. Let T denote the category tree A outputs.
3. R_3 produces an IS $S \subseteq V$, as follows. Let $\hat{\mathcal{C}}$ denote the set of categories that consists of the lowest (closest to the leaves) covering category of every covering branch in T. Let $\hat{Q}$ denote a set of queries constructed by selecting arbitrarily from every category in $\hat{\mathcal{C}}$ a single query that it covers. Observe that $\hat{Q}$ is an r-weak partition. We denote by $\hat{V} = \{v \in V \mid q_v \in \hat{Q}\}$ the set of vertices that corresponds to the queries in $\hat{Q}$, and denote by $\hat{G}$ the sub-hypergraph of G induced by $\hat{V}$. R_3 computes $\hat{G}$ and applies over it the algorithm from Lemma 11, producing an IS S, which is the final output.

For simplicity, we ignore rounding issues, as rounding the parameters to the nearest rational fraction has a negligible effect, and the number of vertices, n, can be manipulated by adding vertices that are connected to all other vertices.

Hardness bounds. The construction above implies the following hardness bounds.

▶ **Theorem 17.** *The $OCT^r(\mathcal{T}_{\alpha,\beta})$ problem, with constant $\alpha \in (0, 1]$ and $\beta > \frac{1}{2}$, cannot be approximated below a $\tilde{\Theta}(n^{\frac{1}{r+1}})$ factor, unless $ZPP = NP$. For $OCT^r(\mathcal{T}_{\alpha,1})$ this is improved to $\tilde{\Theta}(n)$. For $r = 1$ these bounds hold for the assumption $P \neq NP$. Moreover, $OCT^1(\mathcal{T}_{\alpha,1})$ with maximum query size $d = \Theta(1)$ is hard to approximate below a $\tilde{\Theta}(\alpha d)$ factor, unless $BPP = NP$. Lastly, $OCT^1(\mathcal{T}_{1,1})$ is NP-hard even when all queries are of size exactly 3. The bounds for $OCT^1(\mathcal{T}_{\alpha,1})$ hold even when query intersections are of cardinality at most 1.*

We explain below the intuition underlying the reduction and the proof outline.

Intuition. When $\beta = 1$, there are no joint elements, and each query consists of all the relevant edge elements along with padding elements that ensure its size is exactly n^r. In the Exact variant, every covering branch in T covers exactly one query, and this independently-covered set of queries corresponds to a set of vertices that is independent in G. Therefore, if T covers $poly(n)$ queries, we can find an IS of the same size in G.

When relaxing the precision threshold, α, it becomes possible for the same branch to cover multiple queries. As we want to select one query from each branch to ensure independence, it may no longer be the case that the number of covering branches is of the same order as the solution. Nevertheless, on every covering branch, the covering category C closest to the root must contain all elements of all covered queries on the same branch. If there are many such queries, then C would not satisfy the precision requirement. Intuitively, a branch can cover no more than $O(\frac{1}{\alpha}) = O(1)$ queries. It follows that the set of independently-covered queries, $\hat{Q}$, is of the same order as the score of the tree in this case as well.

Matters are more complicated when the recall threshold β is also relaxed. It is no longer the case that an independently-covered set of queries corresponds to an IS. It is now possible for $(r+1)$ such queries to correspond to an edge in G, as the cover of at least one of these queries can avoid containing that edge-element. Without including joint elements in the reduction, the cover of every query could omit a constant fraction of the edge elements, which amounts to $O(n^{r+1})$ edge elements, such that a large independently-covered set of queries could correspond to even a very dense subgraph in G.

To that end, we show that a cover of a query must include a joint element per every omitted edge element. Since all queries share the joint elements, this hinders the ability of covers in other branches to omit edge elements. It follows that the total number of edges in a subgraph corresponding to an independently-covered set of queries contains at most $O(n^r)$ edges which is the total number of joint elements. Therefore, a tree of high score would correspond to a large vertex set which is also sparse. From this "almost IS" we can derive a somewhat smaller, but still polynomial-sized, IS, using Lemma 11.

Adding joint elements may allow covering more queries on a single branch, as including joint elements in a category contributes to its potential cover of all queries. However, we show that the number of covered queries by a single branch is bounded by a constant.

We ensure that the optimal OCT solution is of score $\tilde{\Theta}(n)$. Thus, if the approximation factor of A is low, the eventually derived IS is large. In particular, we ensure that the tree contains a category partition of the same score so that all bounds also hold for OCP. Observe that the maximum IS in G induces the category partition where every category covers a single query pertaining to a vertex in the set, with all covers including all of the non-joint elements and no joint elements. The categories in this partition satisfy the recall condition as narrowly as possible. Intuitively, this construction means that, while joint elements help an algorithm to an extent, beyond that it must make progress on the MIS problem.

5.2 General threshold parameters

We have examined so far the hardness of various special cases of $OCT^r(\mathcal{T}_{\alpha,\beta})$ where the recall threshold is $\beta > \frac{1}{2}$. In particular, we proved for $\frac{1}{2} < \beta < 1$ inapproximability of $\tilde{\Theta}(n^{\frac{1}{r+1}})$. We next devise a more involved construction to show that this result extends to $\beta \leq \frac{1}{2}$ and also provide more general bounds for polynomially small threshold parameters. We note that since the modified construction is randomized, the bound derived for $r = 1$ does not hold under the weaker assumption of $P \neq NP$, unlike in the first construction.

Modifications. To facilitate a precise discussion, we first define, given a subset of queries Q', the multiplicity $M_{Q'}(e) = |\{q \in Q' \mid e \in q\}|$ of an element e in Q' as the number of queries in Q' that e appears in. The reduction used for Theorem 17 becomes ineffective because in the OCT instance constructed by R, the set of joint elements makes up a $(1-\beta)$-fraction of every query, and for $\beta \leq 1/2$ the set of joint elements becomes large enough, such that a category, that consists exactly of this set, covers all queries, yielding the optimal tree score.

To fix this, we need to alter the construction such that joint elements are not shared by all queries. We want to limit the number of joint elements with high multiplicity in any large query set (we will formalize this high-level statement with concrete thresholds in Lemma 21), to make it hard for a single branch to cover it while retaining properties essential for the hardness proof.

Concretely, we want any single joint element to be shared by many queries, and for a $(1-\beta)$-fraction of every query to consist of joint elements, so that the tree that corresponds to the optimal MIS solution narrowly exceeds the recall requirements. This requires using more joint elements. However, having more joint elements can make $\hat{G}$ less sparse, reducing the size of the produced IS. To achieve these desired properties while minimally increasing the number of joint elements, we devise a randomized reduction. Moreover, we parameterize it to efficiently capture sub-constant ranges of α and β, to aim for a slower decay in the hardness bound, as these thresholds are decreased.

Generalized reduction from MIS. Our revised MIS algorithm denoted by R' consists of a sequence of three procedures, R'_1, R'_2 and R'_3. To avoid a convoluted presentation, we reuse some of the notation, initially defined in the context of the first algorithm R.

1. Given an $(r+1)$-uniform hypergraph, $G=(V,E)$, R'_1 transforms it into an instance $Q=\{q_v \mid v \in V\}$ of $OCT^r(\mathcal{T}_{\alpha,\beta})$. Each query, q_v, contains all the edge elements that correspond to edges incident to v in G, and as many unique padding elements as necessary, such that the number of non-joint elements in every query is exactly n^r. Finally, we distribute $(\frac{1}{\beta}-1)\frac{\log^3 n}{\alpha}n^r$ distinct joint elements to queries via the following randomized scheme. We draw uniformly randomly $(\frac{1}{\beta}-1)n^r$ partitions of Q into $\frac{\log^3 n}{\alpha}$ subsets, each of size $\frac{\alpha n}{\log^3 n}$. Let $\hat{\mathbb{P}}$ denote this set of partitions. In every partition, $\hat{p} \in \hat{\mathbb{P}}$, every set, $\hat{s} \in \hat{p}$, in that partition is assigned a distinct joint element to be included in all queries in the set. Note that the size of each query is now exactly $\frac{n}{\beta}$.
2. The procedure R'_2, same as R_2, runs over Q the given $OCT^r(\mathcal{T}_{\alpha,\beta})$ algorithm A with an approximation guarantee factor of γ. Let T denote the category tree A outputs.
3. Finally, R'_3 is the same as R_3, except for the following modification: if there is a branch in T that covers more than $\tilde{\Theta}(\frac{1}{\alpha})$ queries, then it outputs DO NOT KNOW, and otherwise proceeds as R_3 to produce an IS S.

Generalized hardness bounds. We now state the approximation bounds implied by the revised reduction, followed by the intuition underlying the proof.

▶ **Theorem 18.** *The $OCT^r(\mathcal{T}_{\alpha,\beta})$ problem cannot be approximated below a $\tilde{\Theta}((\alpha^{(r+2)}\beta n)^{\frac{1}{r+1}})$ factor, unless $ZPP=NP$.*

Intuition. The most significant component in the proof is the following technical Lemma.

▶ **Lemma 19.** *W.p. $1-o(1)$ (over the choices of partitions in $\hat{\mathbb{P}}$) the maximum number of queries in Q a single branch (in any tree) can cover is $\tilde{O}(\frac{1}{\alpha})$.*

We wish to show that precision cannot be maintained past a certain number of covered queries on a branch. We use the term *relevant cover* of a query q, to refer to the intersection of q with its covering category C, with the *relevant cover size* being $|q \cap C|$. One must be careful in selecting the query for which the precision condition is invoked, to derive a tight bound. On the one hand, we aim to select a query covered close to the root, so that its

covering category contains the covers of many other queries. On the other hand, we want to select a query whose relevant cover is small. Thus, we first prove that for any branch, there exists a query q covered by C, such that at least a constant fraction of the covered queries on the branch are covered by C or a lower category, and that the average relevant cover size of these queries is smaller than the relevant cover size of q by at most a logarithmic factor.

▶ **Lemma 20.** *Given a branch B that covers k' queries, there exists a query q covered by a category C in B, with the following two properties:*

1. *the set of queries, Q_k, covered by C or categories below C is of cardinality $k = \Theta(k')$.*
2. *let $d \in [1, \frac{1}{\beta}]$ denote the value for which the average relevant cover size of queries in Q_k is $n^r d$, then the relevant cover size of q is at most $(2 \log n) n^r d$.*

Given q and C as in Lemma 20, we derive from the precision condition an upper bound on $|C|$. On the other hand, C contains the union of the k covers of the queries in Q_k, and we show that for $k = \tilde{\omega}(\frac{1}{\alpha})$, the union of the k covers, and thereby C must contain many elements, beyond the upper bound, resulting in a contradiction. The key to proving that C contains many elements is bounding the multiplicity of the joint elements in Q_k. If all elements had constant multiplicity, then an α precision threshold implies that, when the relevant covers are on average of roughly the same size as the relevant cover of q (which is the case following Lemma 20), the number of covered queries is $O(\frac{1}{\alpha})$. To that end, we show that the multiplicity of almost every joint element in Q_k does not exceed $\tilde{O}(\frac{k}{\alpha})$.

▶ **Lemma 21.** *For any set Q_k of $k = \omega(\frac{\log^3 n}{\alpha})$ queries, w.p. $1 - o(1)$, there are at most $\frac{n^r}{2}$ partitions in $\hat{\mathbb{P}}$ where a joint element is assigned to more than $\theta = \frac{\alpha k}{8 \log n}$ queries in Q_k.*

The proof of Lemma 21 consists of a combination of probabilistic arguments. We first prove that this θ bound on the number of partitions holds for a uniformly randomly selected set of k queries w.p. $1 - o(n^{-k})$. Then, by using a union bound argument, it will follow that this bound holds for any selection of k queries w.p. $1 - o(1)$.

To prove the bounds of Lemma 21 for a randomly selected set Q_k of k queries, observe that a joint element is shared by polylogarithmically less than a $\frac{1}{\alpha}$-fraction of the queries in Q. Therefore, its expected multiplicity in Q_k would constitute the same fraction. To bound the probability of significantly deviating from this expectation, we show that the multiplicity of any joint element in Q_k is a hypergeometric random variable, and use a tail bound. Following a different union bound argument, this bound on the probability is extended over every joint element assigned in a given partition in $\hat{\mathbb{P}}$. Finally, since the partitions in $\hat{\mathbb{P}}$ are chosen independently, we use a Chernoff bound to derive an upper bound, that holds with high probability, on the number of partitions in $\hat{\mathbb{P}}$ in which a joint element with high multiplicity was assigned. We show that if these deviations occur sufficiently rarely, as stated in Lemma 21, then the cardinality of C increases as a function of k, deriving the bound $k = \tilde{O}(\frac{1}{\alpha})$.

6 Other Variants

So far we have proven hardness of $OCT^r(\mathcal{T}_{\alpha,\beta})$. In this section, we provide approximation hardness bounds for the remaining OCT variants, via reductions from $OCT^r(\mathcal{T}_{\alpha,\beta})$ and Theorems 17 and 18.

We first show that the $\tilde{\Theta}(n^{\frac{1}{r+1}})$ bound of $OCT^r(\mathcal{T}_{\alpha,\beta})$ with constant thresholds extends to the threshold versions of Jaccard and F_1 scores, with similar inapproximability for sub-constant thresholds as well. We then use these results to derive bounds for the cutoff versions of these functions, which only differ for $\delta = o(1)$.

We formulate our proofs schematically, such that they may be applied to threshold and cutoff variants of other functions.

For threshold functions, we derive the following bounds.

▶ **Theorem 22.** *The variants* $OCT^r(\hat{J}_\delta)$ *and* $OCT^r(\hat{F}_{1(\delta)})$ *cannot be approximated below a* $\tilde{\Theta}((\delta^{r+3}n)^{\frac{1}{r+1}})$ *factor, unless* $ZPP = NP$. *For* $r = 1$, *we have* $\tilde{\Theta}(\sqrt{n})$ *inapproximability, assuming* $P \neq NP$, *for* $OCT^r(\hat{J}_\delta)$ *with* $\delta > \frac{1}{2}$ *and* $OCT^r(\hat{F}_{1(\delta)})$ *with* $\delta > \frac{2}{3}$.

Finally, we provide bounds for cutoff functions, that follow from Theorem 22.

▶ **Theorem 23.** *The variants* $OCT^r(\bar{J}_\delta)$ *and* $OCT^r(\bar{F}_{1(\delta)})$, *with* $\delta \in [0,1)$, *have* $\tilde{\Theta}((\delta^{2r+4}n)^{\frac{1}{r+1}})$ *inapproximability, unless* $ZPP = NP$. *For* $r = 1$, *we have* $\tilde{\Theta}(\sqrt{n})$ *inapproximability, assuming* $P \neq NP$, *for* $OCT^r(\bar{J}_\delta)$ *with* $\delta > \frac{1}{2}$ *and* $OCT^r(\bar{F}_{1(\delta)})$ *with* $\delta > \frac{2}{3}$.

7 Related Work

The construction of category trees/taxonomies has been studied in multiple domains, including e-commerce, document management, and question answering [9, 24, 12]. Many algorithms have been devised for automating taxonomy construction [17, 9, 18] and maintenance, [22, 24, 26] employing different clustering approaches [9, 17], as well as crowdsourcing [21].

In the lines of work specified above, the quality of the resulting taxonomy is assessed along the following two dimensions.

The first dimension of quality assessment is user-study [9, 17], an evaluation which we incorporate w.r.t. our model in the complementary empirical work [3]. This evaluation is naturally entirely subjective.

The second dimension, which is the focus of the present paper, is the similarity of the resulting category tree to a given (combinatorially unrestricted) ground-truth set of items/documents. For example, the F_1 score used in [17, 9, 18] is a variant (without a threshold) of our corresponding F_1 measure for $r = 1$. Similarly, [22] computes recall and F_1 scores for the resulting trees, also with $r = 1$.

To our knowledge, however, no previous work investigates the theoretical complexity of the optimization problem of computing the tree of the highest score. The score is only used as an evaluation measure, to which the algorithm is oblivious. This approach, to an extent, is loosely justified by our worst-case bounds. Nevertheless, we show in [3] and [4], that leveraging the relation we outlined in Section 4 to the weighted MIS problem, allows solving well (and, in some cases, optimally) real-world problem instances, via extensively studied MIS solvers.

Our model differs from clustering models [19, 13] that typically focus on item-similarity, optimizing the similarity within each cluster or the dissimilarity across clusters. Moreover, these models are commonly defined by pairwise similarities, while our model also considers relations of a higher order. Thus, closest to our work in this domain is the field of hypergraph partitioning (clustering) [14, 16]. Specifically, the OCP problem with copy-bound $r = 1$ corresponds to seeking a partition of the vertices that maximizes the weight of (hyper)edges for which there is a similar set in the partition. Relaxing the copy-bound corresponds to overlapping clusters. Importantly, this relation between hypergraph clustering and our model is different from the more artificial relation leveraged in our reductions, where we cluster the hypergraph edges, instead of the vertices. Nevertheless, our proposed framework differs from existing models in several aspects. Notably, hypergraph clustering typically studies a

multi-way cut problem, intending to minimize the weight of the cut edges. Recently [16] suggested that there is a benefit in quantifying how an edge is cut, in terms of which subsets of its vertices are clustered together. Our work is relevant in that respect, as we quantify how similar these subsets are to the original edge.

A work resembling ours in a different aspect is [25], where the objective is to maximize the edge weights inside the cluster (we also maximize the covered "demand", instead of the less natural minimization of uncovered demand). However, the models of [16] and [25] (and many others [15, 8]) are easier to approximate, due to principal technical differences (e.g., bounds on the size and number of clusters), and we are not aware of clustering research that resembles our model or bounds.

8 Conclusion

In this paper, we studied the hardness of computing categorizations with a bounded number of possible repetitions, that best capture a given collection of item sets. We defined a model that captures various practical settings and proved inapproximability results for multiple variants and special cases. We also provided an algorithm for the Exact variant with an approximation guarantee that depends on finer input parameters.

An interesting direction for future work would be to identify more special cases that admit improved performance. Another intriguing avenue of exploration is determining for cases where we showed $\tilde{\Theta}(\sqrt{n})$ hardness, whether one can devise algorithms with matching approximation guarantees or prove stronger bounds.

References

1 https://export.ebay.com/en/start-sell/selling-basics/seller-fees/fees-optional-listing-upgrades/.

2 Rakesh Agrawal, Amit Somani, and Yirong Xu. Storage and querying of e-commerce data. In *VLDB*, pages 149–158, 2001.

3 Uri Avron, Shay Gershtein, Ido Guy, Tova Milo, and Slava Novgorodov. Category Tree Construction from Search Queries in E-Commerce. *https://slavanov.com/research/concat_tr.pdf*.

4 Uri Avron, Shay Gershtein, Ido Guy, Tova Milo, and Slava Novgorodov. ConCaT: Construction of Category Trees from Search Queries in E-Commerce. In *ICDE*, 2021.

5 Slobodan Beliga, Ana Meštrović, and Sanda Martinčić-Ipšić. An overview of graph-based keyword extraction methods and approaches. *JIOS*, 39(1):1–20, 2015.

6 Amey Bhangale and Subhash Khot. UG-hardness to NP-hardness by Losing Half. In *CCC*, 2019.

7 Yair Caro and Zsolt Tuza. Improved lower bounds on k-independence. *Journal of Graph Theory*, 15(1):99–107, 1991.

8 Karthekeyan Chandrasekaran, Chao Xu, and Xilin Yu. Hypergraph k-cut in randomized polynomial time. *Mathematical Programming*, pages 1–29, 2019.

9 Shui-Lung Chuang and Lee-Feng Chien. A practical web-based approach to generating topic hierarchy for text segments. In *CIKM*, page 127–136, 2004.

10 Shay Gershtein, Uri Avron, Ido Guy, Tova Milo, and Slava Novgorodov. On the Hardness of Category Tree Construction (full). *https://slavanov.com/research/icdt22_full.pdf*.

11 Thomas Hofmeister and Hanno Lefmann. Approximating maximum independent sets in uniform hypergraphs. In *Proc. of MFCS*, pages 562–570, 1998.

12 Yi-Hsiang Hsieh, Shih-Hung Wu, Liang-Pu Chen, and Ping-Che Yang. Constructing hierarchical product categories for e-commerce by word embedding and clustering. In *IRI*, pages 397–402, 2017.

13 Anna Huang. Similarity measures for text document clustering. In *NZCSRSC*, volume 4, pages 9–56, 2008.

14 George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: applications in VLSI domain. *VLSI*, 7(1):69–79, 1999.

15 Tom Leighton, Fillia Makedon, and SG Tragoudas. Approximation algorithms for VLSI partition problems. In *ISCAS*, pages 2865–2868, 1990.

16 Pan Li and Olgica Milenkovic. Inhomogeneous hypergraph clustering with applications. In *NIPS*, pages 2308–2318, 2017.

17 Kunal Punera, Suju Rajan, and Joydeep Ghosh. Automatically learning document taxonomies for hierarchical classification. In *Proc. of WWW*, 2005.

18 Cécile Robin, James O'Neill, and Paul Buitelaar. Automatic taxonomy generation - a use-case in the legal domain, 2017. `arXiv:1710.01823`.

19 Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

20 Matthew Skala. Hypergeometric tail inequalities: ending the insanity, 2013. `arXiv:1311.5939`.

21 Yuyin Sun, Adish Singla, Dieter Fox, and Andreas Krause. Building hierarchies of concepts via crowdsourcing. *CoRR*, abs/1504.07302, 2015. `arXiv:1504.07302`.

22 Lei Tang, Jianping Zhang, and Huan Liu. Acclimatizing taxonomic semantics for hierarchical content classification. In *Proc. of KDD*, pages 384–393, 01 2006.

23 Eli Upfal. *Probability and computing: randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

24 Quan Yuan, Gao Cong, Aixin Sun, Chin-Yew Lin, and Nadia Magnenat Thalmann. Category hierarchy maintenance: a data-driven approach. In *SIGIR*, pages 791–800, 2012.

25 Wenxing Zhu and Chuanyin Guo. Local search approximation algorithms for the complement of the min-k-cut problems. 2010.

26 Hai Zhuge and Lei He. Automatic maintenance of category hierarchy. *Future Generation Computer Systems*, 67:1 – 12, 2017.

27 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proc. of STOC*, pages 681–690, 2006.

Linear Programs with Conjunctive Queries

Florent Capelli
Univ. Lille, Inria, CNRS, UMR 9189 - CRIStAL, F-59000 Lille, France

Nicolas Crosetti
Univ. Lille, Inria, CNRS, UMR 9189 - CRIStAL, F-59000 Lille, France

Joachim Niehren
Univ. Lille, Inria, CNRS, UMR 9189 - CRIStAL, F-59000 Lille, France

Jan Ramon
Univ. Lille, Inria, CNRS, UMR 9189 - CRIStAL, F-59000 Lille, France

Abstract

In this paper, we study the problem of optimizing a linear program whose variables are the answers to a conjunctive query. For this we propose the language LP(CQ) for specifying linear programs whose constraints and objective functions depend on the answer sets of conjunctive queries. We contribute an efficient algorithm for solving programs in a fragment of LP(CQ). The naive approach constructs a linear program having as many variables as there are elements in the answer set of the queries. Our approach constructs a linear program having the same optimal value but fewer variables. This is done by exploiting the structure of the conjunctive queries using generalized hypertree decompositions of small width to factorize elements of the answer set together. We illustrate the various applications of LP(CQ) programs on three examples: optimizing deliveries of resources, minimizing noise for differential privacy, and computing the s-measure of patterns in graphs as needed for data mining.

2012 ACM Subject Classification Theory of computation → Logic and databases

Keywords and phrases Database queries, linear programming, hypergraph decomposition

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.5

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-01981553`

Funding This work was partially supported by the French Agence Nationale de la Recherche, AGGREG project reference ANR-14-CE25-0017-01, Headwork project reference ANR-16-CE23-0015 and by a grant of the Conseil Régional Hauts-de-France. The project DATA, Ministère de l'Enseignement Supérieur et de la Recherche, Région Nord-Pas de Calais and European Regional Development Fund (FEDER) are acknowledged for supporting and funding this work.

Acknowledgements We also thank Sylvain Salvati, Sophie Tison and Yuyi Wang for fruitful discussions and anonymous reviewers of a previous version of this paper for their helpful comments.

1 Introduction

When modeling optimization problems it often seems natural to separate the logical constraints from the relational data. This holds for linear programming with AMPL [5] and for constraint programming in MiniZinc [15]. It was also noticed in the context of database research, when using integer linear programming for finding optimal database repairs as proposed by Kolaitis, Pema and Tan [12], or when using linear optimization to explain the result of a database query to the user as proposed by Meliou and Suciu [14]. Moreover, tools like `SolveDB` [19] have been developed to better integrate mixed integer programming and thus linear programming into relational databases.

We also find it natural to define the relational data of linear optimization problems by database queries. For this reason, we propose the language of linear programs with conjunctive queries $LP(CQ)$ in the present paper. The objective is to become able to specify weightings of answer sets of database queries, that optimize a linear objective function subject to linear constraints. The optimal weightings of $LP(CQ)$ programs can be computed in a naive manner, by first answering the database queries, and then solving a linear program parametrized by the answer sets. We then approach the question – to our knowledge for the first time – of whether this can be done with lower complexity for subclasses of conjunctive queries such as the class of acyclic conjunctive queries.

As our main contribution we present a more efficient algorithm for computing the optimal value of a program in the fragment of so-called projecting $LP(CQ)$ programs for which we also bound the hypertree width of the queries. The particular case of width 1 covers the class of acyclic conjunctive queries. By using hypertree decompositions, our algorithm is based on a factorized interpretation of a projecting $LP(CQ)$ program over a database. The factorized interpretation uses other linear program variables, that represent sums of the linear program variables in the naive interpretation. The number of linear program variables in the factorized interpretation depends only on the widths of the hypertree decompositions of the queries in the $LP(CQ)$ program, rather than on the number of query variables. In this manner, our more efficient algorithm can decrease the data complexity, i.e., the degree of the polynomial in the upper bound of the run time of the naive algorithm. With respect to the combined complexity, the special case of projecting $LP(CQ)$ programs with bounded quantifier depth becomes tractable for acyclic conjunctive queries, while it is NP-complete in general.

We prove the correctness of the factorized interpretation with respect to the naive interpretation. For this we have to show a correspondence between weightings of answer sets on the naive interpretation, and weightings of answer sets on the factorized interpretation. This correspondence can be seen as an independent contribution as it shows that one can reconstruct a relevant weighting of the answer set of a quantifier free conjunctive query by only knowing the value of the projected weighting on the bags of the tree decomposition.

Conjunctive queries with existential quantifiers are dealt with by showing that one can find an equivalent projecting $LP(CQ)$ program with quantifier free conjunctive queries only.

1.1 Applications

A wide range of applications of linear programs can benefit from conjunctive queries.

Resource Delivery Optimization. We consider a situation in logistics where a company received orders for specific quantities of resource objects. The objects must be produced at a factory, then transported to a warehouse before being delivered to the buyer. The objective is to fulfill every order while minimizing the overall delivery costs and respecting the production capacities of the factories as well as the storing capacities of the warehouses.

Let F be the set of factories, O the set of objects, W the set of warehouses and B the set of buyers. We consider a database $\mathbb{D}$ with elements in the domain $D = \mathsf{F} \uplus \mathsf{O} \uplus \mathsf{W} \uplus \mathsf{B} \uplus \mathbb{R}_+$. The elements $d \in D$ encoding a positive real number can be decoded back by applying the database's functions $\mathbf{num}^{\mathbb{D}}$, yielding the positive real number $\mathbf{num}^{\mathbb{D}}(d) \in \mathbb{R}_+$. The database $\mathbb{D}$ has four tables. The first table $prod^{\mathbb{D}} \subseteq \mathsf{F} \times \mathsf{O} \times \mathbb{R}_+$ contains triples (f, o, q) stating that the factory f can produce up to q units of object o. The second table $order^{\mathbb{D}} : \mathsf{B} \times \mathsf{O} \times \mathbb{R}_+$ contains triples (b, o, q) stating that the buyer b orders q units of object o. The third table $store^{\mathbb{D}} \subseteq \mathsf{W} \times \mathbb{R}_+$ contains pairs (w, l) stating that the warehouse w has a storing limit of l.

$$
\begin{aligned}
&\textbf{minimize}\\
&\quad \textstyle\sum_{(f,w,c):route(f,w,c)} \textbf{num}(c)\ \textbf{weight}_{(f',w',b',o'):f'\doteq f\wedge w'\doteq w}(Q)\\
&\quad +\textstyle\sum_{(w,b,c):route(w,b,c)} \textbf{num}(c)\ \textbf{weight}_{(f',w',b',o'):w'\doteq w\wedge b'\doteq b}(Q)\\
&\textbf{subject to}\\
&\quad \forall(f,o,q){:}prod(f,o,q).\textbf{weight}_{(f',w',b',o'):f'\doteq f\wedge o'\doteq o}(Q) \leq \textbf{num}(q)\\
&\wedge\ \forall(b,o,q){:}order(b,o,q).\textbf{weight}_{(f',w',b',o'):b'\doteq b\wedge o'\doteq o}(Q) \geq \textbf{num}(q)\\
&\wedge\ \forall(w,l){:}store(w,l).\textbf{weight}_{(f',w',b',o'):w'\doteq w}(Q) \leq \textbf{num}(l)
\end{aligned}
$$

Figure 1 A $LP(CQ)$ program for the resource delivery optimization where $Q = dlr(f', w', b', o')$.

The fourth table $route^{\mathbb{D}} : (\mathsf{F} \times \mathsf{W} \times \mathbb{R}_+) \cup (\mathsf{W} \times \mathsf{B} \times \mathbb{R}_+)$ contains triples (f, w, c) stating that the transport from factory f to warehouse w costs c, and triples (w, b, c) stating that the transport from warehouse w to buyer b costs c. The query:

$$dlr(f, w, b, o) = \exists q.\exists q_2.\exists c\exists c_2.\ prod(f, o, q) \wedge order(b, o, q_2) \wedge route(f, w, c) \wedge route(w, b, c_2)$$

selects from the database $\mathbb{D}$ all tuples (f, w, b, o) such that the factory f can produce some objects o to be delivered to buyer b through the warehouse w. Let $Q = dlr(f', w', b', o')$. The goal is to determine for each of these possible deliveries the quantity of the object that should actually be sent. These quantities are modelled by the unknown weights θ^{α}_{Q} of the query answers $\alpha \in sol^{\mathbb{D}}(Q)$. For any factory f and warehouse w the sum $\sum_{\alpha\in sol^{\mathbb{D}}(Q\wedge w'\doteq w\wedge f'\doteq f)} \theta^{\alpha}_{Q}$ is described by the expression $\textbf{weight}_{(f',w',b',o'):f'\doteq f\wedge w'\doteq w}(Q)$ when interpreted over $\mathbb{D}$.

We use the $LP(CQ)$ program in Figure 1 to describe the optimal weights that minimize the overall delivery costs. The weights depend on the interpretation of the program over the database, since $\mathbb{D}$ specifies the production capacities of the factories, the stocking limits of the warehouses, etc. The program has the following constraints:

- for each $(f, o, q) \in prod^{\mathbb{D}}$ the overall quantity of object o produced by f is at most q.
- for each $(b, o, q) \in order^{\mathbb{D}}$ the overall quantity of objects o delivered to b is at least q.
- for each $(w, l) \in store^{\mathbb{D}}$ the overall quantity of objects stored in w is at most l.

By answering the query Q on the database $\mathbb{D}$ and introducing a linear program variable θ^{α}_{Q} for each of the query answers α, we can interpret the $LP(CQ)$ program in Figure 1 as a linear program. However the number of answers of Q and thus the number of variables in this program could be cubic in the size of the database, which quickly grows too big.

Our factorized interpretation for the projecting $LP(CQ)$ program in Figure 1 produces a linear program that only has a quadratic number of variables since the query Q (as well as the whole $LP(CQ)$ program) has a hypertree decomposition of width 2.

Minimizing Noise for ε-Differential Privacy. The strategy of differential privacy is to add noise to the relational data before publication. Roughly speaking, the general objective of ε-differential privacy [4] is to add as little noise as possible, without disclosing more than an ε amount of information. We illustrate this with the example of a set of hospitals which publish medical studies aggregating results of tests on patients, which are to be kept confidential. We consider the problem of how to compute the optimal amount of noise to be added to each separate piece of sensitive information (in terms of total utility of the studies) while guaranteeing ε-differential privacy. We show that this question can be solved (approximately) by computing the optimal solution of a projecting program in $LP(CQ)$

with a single conjunctive query that is acyclic, i.e., of hypertree with 1. While the naive interpretation yields a linear program with a quadratic number of variables in the size of the database, the factorized interpretation requires only a linear number. The example is worked out in the full version [2].

Computing the s-Measure for Graph Pattern Matching. A matching of a subgraph pattern in a graph is a graph homomorphism from the pattern to the graph. The s-measure of Wang et al. [20] is used in data mining to measure the frequency of matchings of subgraph patterns, while accounting for overlaps of different matchings. The idea is to find a maximal weighting for the set of matchings, such that for any node of the subgraph pattern, the set of matchings mapping it on the same graph node must have an overall weight less than 1. This optimization problem can be expressed by a projecting $LP(CQ)$ program over a database storing the graph. The conjunctive query of this program expresses the matching of the subgraph pattern. The hypertree width of this conjunctive query is bounded by the hypertree width of the subgraph pattern. Our factorized interpretation therefore reduces the size of the linear program for subgraph patterns with small hypertree width. More information on the $LP(CQ)$ program can be found in the full version [2].

1.2 Related Work

Our result builds on well-known techniques using dynamic programming on tree decompositions of the hypergraph of conjunctive queries. These techniques were first introduced by Yannkakis [21] who observed that so-called acyclic conjunctive queries could be answered in linear time using dynamic programming on a tree whose nodes are in correspondence with the atoms of the query. Generalizations have followed in two directions: on the one hand, generalizations of acyclicity such as notions of hypertree width [6, 7, 8] have been introduced and on the other hand enumeration and aggregation problems have been shown to be tractable on these families of queries such as finding the size of the answer set [18] or enumerating it with small delay [1]. These tractability results can be obtained in a unified and generalized way by using factorized databases introduced by Olteanu and Závodný [16, 17], from which our work is inspired. Factorized databases provide succinct representations for answer sets of queries on databases. The representation enjoys interesting syntactic properties allowing to efficiently solve numerous aggregation problems on answer sets in polynomial time in the size of the representation. Olteanu and Závodný [17] have shown that when the fractional hypertree width of a query Q is bounded, then one can construct, given a hypertree decomposition of Q and a database $\mathbb{D}$, a factorized databases representing the answers of Q on $\mathbb{D}$ of polynomial size. They also give a $O(1)$ delay enumeration algorithm on factorized databases. Combining both results gives a generalization of the result of Bagan, Durand and Grandjean [1] on the complexity of enumerating the answers of conjunctive queries.

Our result heavily draws inspiration from this approach as we use bottom up dynamic programming on hypertree decomposition of the input query Q to construct a partial representation of the answers set of Q on database $\mathbb{D}$ that we later use to construct a factorized interpretation of the linear program to solve. While our approach could be made to work directly on factorized representations of queries answer sets as defined by Olteanu and Závodný [17], we choose to directly work on tree decompositions because we need to incorporate some structure of the linear program into our tree decomposition to efficiently handle complex linear programs.

Linear expressions	$S, S' \in L_{\text{E}}$	$::=$	$c \mid \xi \mid cS \mid S + S'$
Linear constraints	$C, C' \in L_{\text{C}}$	$::=$	$S \leq S' \mid C \wedge C' \mid \mathit{true}$
Linear programs	$L \in L_{\text{P}}$	$::=$	**maximize** S **subject to** C

Figure 2 The set of linear programs L_{P} with variables $\xi \in \Xi$ and constants $c \in \mathbb{R}$.

Organization of the paper. Section 2 contains the necessary definitions to understand the paper. Section 3 presents the language $LP(CQ)$ of linear programs parametrized by conjunctive queries and gives its semantics. Section 4 defines a fragment of $LP(CQ)$ for which we propose a more efficient algorithm. Finally, Section 5 presents encouraging practical results on solving the delivery optimization problem using this algorithm. Due to space limit, most proofs and full details on applications to differential privacy and s-measure computation can be found in the full version [2].

2 Preliminaries

Sets, Functions and Relations. Let $\mathbb{B} = \{0, 1\}$ be the set of Booleans, $\mathbb{N}$ the set of natural numbers including 0, $\mathbb{R}_+$ be the set of positive reals including 0 and subsuming $\mathbb{N}$, and $\mathbb{R}$ the set of all reals.

Given any set S and $n \in \mathbb{N}$ we denote by S^n the set of all n-tuples over S and by $S^* = \cup_{n\in\mathbb{N}} S^n$ the set of all words over S. A *weighting* on S is a (total) function $f : S \to \mathbb{R}_+$.

Given a set of (total) functions $A \subseteq D^S = \{f \mid f : S \to D\}$ and a subset $S' \subseteq S$, we define the set of restrictions $A_{|S'} = \{f_{|S'} \mid f \in A\}$. For any binary relation $R \subseteq S \times S$, we denote its transitive closure by $R^+ \subseteq S \times S$ and the reflexive transitive closure by $R^* = R^+ \cup \{(s, s) \mid s \in S\}$.

Variable assignments. We fix a countably infinite set of (query) variables $\mathcal{X}$. For any set D of database elements, an assignment of (query) variables to database elements is a function $\alpha : X \to \mathsf{D}$ that maps elements of a finite subset of variables $X \subseteq \mathcal{X}$ to values of D. For any two sets of variable assignments $A_1 \subseteq \mathsf{D}^{X_1}$ and $A_2 \subseteq \mathsf{D}^{X_2}$ we define their join $A_1 \bowtie A_2 = \{\alpha_1 \cup \alpha_2 \mid \alpha_1 \in A_1, \alpha_2 \in A_2, \alpha_{1|I} = \alpha_{2|I}\}$ where $I = X_1 \cap X_2$.

We also use a few vector notations. Given a vector of variables $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ we denote by $\mathit{set}(\mathbf{x}) = \{x_1, \ldots, x_n\}$ the set of the elements of $\mathbf{x}$. For any variable assignment $\alpha : X \to D$ with $\mathit{set}(\mathbf{x}) \subseteq X$ we denote the application of the assignment α on $\mathbf{x}$ by $\alpha(\mathbf{x}) = (\alpha(x_1), \ldots, \alpha(x_n))$.

Linear programs. Let Ξ be a set of linear program variables. In Figure 2, we recall the definition of the sets of linear expressions L_{E}, linear constraints L_{C}, and linear programs L_{P} with variables in Ξ. We consider the usual linear equations $S \doteq S'$ as syntactic sugar for the constraints $S \leq S' \wedge S' \leq S$. For any linear program $L =$ **maximize** S **subject to** C we call S the objective function of L and C the constraint of L. Note that the linear program **minimize** S **subject to** C can be expressed by **maximize** $-1\ S$ **subject to** C up to negation.

The formal semantics of linear programs is recalled in the full version [2]. Since we will only be interested in variables for positive real numbers – and do not want to impose positivity constraints all over – we restrict variables of linear programs to always be positive

$$\begin{array}{lll} \text{Expressions} & E_1, \ldots, E_n \in Ex_{\mathcal{C}} & ::= x \mid a \\ \text{Conjunctive queries} & Q, Q' \in C\mathrm{Q}_\Sigma & ::= E_1 \doteq E_2 \mid r(E_1, \ldots, E_n) \mid Q \wedge Q' \mid \exists x.Q \mid \mathit{true} \end{array}$$

Figure 3 The set of conjunctive queries $C\mathrm{Q}_\Sigma$ with schema $\Sigma = ((\mathcal{R}^{(n)})_{n\in\mathbb{N}}, \mathcal{C})$ where $x \in \mathcal{X}$, $a \in \mathcal{C}$, and $r \in \mathcal{R}^{(n)}$.

real numbers. For any weightings $\omega : \Xi \to \mathbb{R}_+$, the value of a sum $S \in L\mathrm{E}$ is the real number $[\![S]\!]_\omega \in \mathbb{R}$, and the value of a constraint $C \in L\mathrm{C}$ is the truth value $[\![C]\!]_\omega \in \mathbb{B}$. The optimal solution $[\![L]\!] \in \mathbb{R}$ of a linear program L with objective function S and constraint C is $[\![L]\!] = \max\{[\![S]\!]_\omega \mid \omega : \Xi \to \mathbb{R}_+,\ [\![C]\!]_\omega = 1\}$. It is well-known that the optimal solution of a linear program can be computed in polynomial time [10].

Rooted trees. A digraph is a pair $(\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge sets $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. A digraph is acyclic if there is no $v \in \mathcal{V}$ for which $(v, v) \in \mathcal{E}^+$. For any node $u \in \mathcal{V}$, we denote by $\downarrow u = \{v \in \mathcal{V} \mid (u, v) \in \mathcal{E}^*\}$ the set of nodes in $\mathcal{V}$ reachable over some downwards path from u, and by $\uparrow u = (\mathcal{V} \setminus \downarrow u) \cup \{u\}$ the context of u. A *rooted tree* is an acyclic digraph where $(u, v), (u', v) \in \mathcal{E}$ implies $u = u'$, and there exists a node $r \in \mathcal{V}$ such that $\mathcal{V} = \downarrow r$. In this case, r is unique and called the root of the tree. Observe that in this tree, the paths are oriented from the root to the leaves of the tree.

Relational Databases. A *database schema* is a pair $\Sigma = (R, \mathcal{C})$ where $\mathcal{C}$ a finite set of constants ranged over by a, b and $R = \cup_{n\in\mathbb{N}}\mathcal{R}^{(n)}$ is a finite set of relation symbols. The elements $r \in \mathcal{R}^{(n)}$ are called relation symbols of arity $n \in \mathbb{N}$.

A *database* $\mathbb{D} \in db_\Sigma$ is a tuple $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}})$, where Σ is a schema, D a finite set of database elements, and $r^{\mathbb{D}} \subseteq D^n$ a relation for any relation symbol $r \in \mathcal{R}^{(n)}$ and $a^{\mathbb{D}} \in D$ a database element for any constant $a \in \mathcal{C}$. We also define the database's domain $\mathit{dom}(\mathbb{D}) = D$.

A *database with real numbers* is a tuple $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}}, \mathbf{num}^{\mathbb{D}})$ such that $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}})$ is a relational database and $\mathbf{num}^{\mathbb{D}}$ a partial function from D to $\mathbb{R}$.

Conjunctive Queries. In Figure 3 we recall the notion of conjunctive queries on relational databases. An expression $E \in Ex_{\mathcal{C}}$ is either a (query) variable $x \in \mathcal{X}$ or a constant $a \in \mathcal{C}$. The set of conjunctive queries $Q \in C\mathrm{Q}_\Sigma$ is built from equations $E_1 \doteq E_2$, atoms $r(E_1, \ldots, E_n)$, the logical operators of conjunction $Q \wedge Q'$ and existential quantification $\exists x.Q$. Given a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ and a query Q, we write $\exists \mathbf{x}.Q$ instead of $\exists x_1. \ldots . \exists x_n.Q$.

The set of free variables $\mathit{fv}(Q) \subseteq \mathcal{X}$ are those variables that occur in Q outside the scope of an existential quantifier. A conjunctive query Q is said to be *quantifier free* if it does not contain any existential quantifier.

For any conjunctive query $Q \in C\mathrm{Q}_\Sigma$, set $X \supseteq \mathit{fv}(Q)$ and database $\mathbb{D} \in db_\Sigma$ we define the answer set $\mathit{sol}^{\mathbb{D}}_X(Q)$. It contains all those assignments $\alpha : X \to \mathit{dom}(\mathbb{D})$ for which Q becomes true on $\mathbb{D}$. We also write $\mathit{sol}^{\mathbb{D}}(Q)$ instead of $\mathit{sol}^{\mathbb{D}}_{\mathit{fv}(Q)}(Q)$. Observe that $\mathit{sol}^{\mathbb{D}}(\exists \mathbf{x}.Q) = \mathit{sol}^{\mathbb{D}}(Q)_{|\mathit{fv}(Q)\setminus \mathit{set}(\mathbf{x})}$.

Hypertree Decompositions. Hypertree decompositions of conjunctive queries are a way of laying out the structure of a conjunctive query in a tree. It allows to solve many aggregation problems (such as checking the existence of a solution, counting or enumerating the solutions etc.) on quantifier free conjunctive queries in polynomial time where the degree of the polynomial is given by the width of the decomposition.

▶ **Definition 1.** *Let $X \subseteq \mathcal{X}$ be a finite set of variables. A decomposition tree T of X is a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{B})$ such that:*

- *$(\mathcal{V}, \mathcal{E})$ is a finite directed rooted tree with edges from the root to the leaves,*
- *the bag function $\mathcal{B} : \mathcal{V} \to 2^X$ maps nodes to subsets of variables in X,*
- *for all $x \in X$ the subset of nodes $\{u \in \mathcal{V} \mid x \in \mathcal{B}(u)\}$ is connected in the tree $(\mathcal{V}, \mathcal{E})$,*
- *each variable of X appears in some bag, that is $\bigcup_{u \in \mathcal{V}} \mathcal{B}(u) = X$.*

Now a hypertree decomposition of a quantifier free conjunctive query is a decomposition tree where for each atom of the query there is at least one bag that covers its variables.

▶ **Definition 2** (Hypertree width of quantifier free conjunctive queries)**.** *Let $Q \in C_{\mathrm{Q}_\Sigma}$ be a quantifier free conjunctive query. A* generalized hypertree decomposition of Q *is a decomposition tree $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of $fv(Q)$ such that for each atom $r(\mathbf{x})$ of Q there is a vertex $u \in \mathcal{V}$ such that $set(\mathbf{x}) \subseteq \mathcal{B}(u)$. The* width *of T with respect to Q is the minimal number k such that every bag of T can be covered by the variables of k atoms of Q. The* generalized hypertree width of a query Q *is the minimal width of a tree decomposition of Q.*

We call a conjunctive query α-acyclic if it has general hypertree width 1. The query $r(x, y) \wedge r(y, z)$ has the generalized hypertree decomposition $(\mathcal{V}, \mathcal{E}, \mathcal{B})$ with $\mathcal{V} = \{1, 2, 3\}$, $\mathcal{E} = \{(1, 2), (1, 3)\}$, and $\mathcal{B} = [1/\{y\}, 2/\{x, y\}, 3/\{y, z\}]$ of width 1, so it is α-acyclic.

Many problems can be solved efficiently on conjunctive queries having a small hypertree width. We will mainly be interested in the problem of efficiently computing $sol^{\mathbb{D}}(Q)$.

▶ **Lemma 3.** *Given a tree decomposition $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of a quantifier free conjunctive query $Q \in C_{\mathrm{Q}_\Sigma}$ of width k and a database $\mathbb{D} \in db_\Sigma$, one can compute the collection of bag projections $(sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)})_{u \in \mathcal{V}}$ in time $O((|\mathbb{D}|^k \log(|\mathbb{D}|)) \cdot |T|)$.*

Lemma 3 is folklore: it can be proven by computing the semi-join of every bag in a subtree in a bottom-up fashion, as it is done in [13, Theorem 6.25]. This yields a superset S_u of $sol^{\mathbb{D}}(Q)|_{\mathcal{B}(u)}$ for every u. Then, with a second top-down phase, one can remove tuples from S_u that cannot be extended to a solution of $sol^{\mathbb{D}}(Q)$.

Note that if Q contains n atoms, $sol^{\mathbb{D}}(Q)$ may be of size $O(|\mathbb{D}|^n)$ while $(sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)})_{u \in \mathcal{V}}$ has size $O(|\mathbb{D}|^k \cdot |T|)$ where k is the width of T. In the particular case of α-acyclic conjunctive queries, where $k = 1$, the overall size of the projections is linear. It gives a succinct way of describing the set of solutions of Q that we exploit in this paper.

We observe here that Lemma 3 is still valid if one replaces hypertree width with the more general notion of fractional hypertree width [9]. Since our tractability results only follows from the bounds presented in Lemma 3, it implies that our results also holds for tree decomposition having bounded fractional hypertree width. We however choose to present our result by using hypertree width since it is easier to define.

Parts of our result will be easier to describe on so-called normalized decomposition trees:

▶ **Definition 4.** *Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree. We call a node $u \in \mathcal{V}$ of T:*
- **an extend node** *if it has a single child u' and $\mathcal{B}(u) = \mathcal{B}(u') \cup \{x\}$ for some $x \in \mathcal{X} \setminus \mathcal{B}(u')$,*
- **a project node** *if it has a single child u' and $\mathcal{B}(u) = \mathcal{B}(u') \setminus \{x\}$ for some $x \in \mathcal{X} \setminus \mathcal{B}(u)$,*
- **a join node** *if it has $k \geq 1$ children $u_1, ..., u_k$ with $\mathcal{B}(u) = \mathcal{B}(u_1) = ... = \mathcal{B}(u_k)$.*

We call T normalized [1] *if all its nodes in $\mathcal{V}$ are either extend nodes, project nodes, join nodes, or leaves.*

[1] In the literature this property is referred to as "nice" tree decompositions.

Constant numbers	$N \in Num_{\mathcal{C}}$	::=	$c \mid \mathbf{num}(E)$
$LP(CQ)$ expressions	$S, S' \in L_{\mathrm{E}\Sigma}$	::=	$\mathbf{weight}_{\mathbf{x}:Q'}(Q) \mid \sum_{\mathbf{x}:Q} S \mid NS \mid S + S' \mid N$
$LP(CQ)$ constraints	$C, C' \in L_{\mathrm{C}\Sigma}$	::=	$S \leq S' \mid C \wedge C' \mid \mathit{true} \mid \forall \mathbf{x}{:}Q.C$
$LP(CQ)$ programs	$L \in L_{\mathrm{P}\Sigma}$	::=	$\mathbf{maximize}\ S\ \mathbf{subject\ to}\ C$
			where $fv(S) = fv(C) = \emptyset$.

Figure 4 The set of $LP(CQ)$ programs $L_{\mathrm{P}\Sigma}$ where $c \in \mathbb{R}$, $E \in Ex_{\mathcal{C}}$, $\mathbf{x} \in \mathcal{X}^*$ and $Q, Q' \in C_{\mathrm{Q}\Sigma}$.

It is well-known that tree decompositions can always be normalized without changing the width. Thus normalization does not change the asymptotic complexity of the algorithms.

▶ **Lemma 5** (Lemma of 13.1.2 of [11]). *For every tree decomposition of $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of Q of width k, there exists a normalized tree decomposition $T' = (\mathcal{V}', \mathcal{E}', \mathcal{B}')$ having width k. Moreover, one can compute T' from T in polynomial time.*

3 Linear Programs with Conjunctive Queries

We want to assign weights to the answers of a conjunctive query on a database, such that they maximize a linear objective function subject to linear constraints. For this, we introduce the language $LP(CQ)$ of linear programs with conjunctive queries.

3.1 Syntax

Its syntax is given in Figure 4. Note that an example of an $LP(CQ)$ program for optimal warehouse selection was already given in Figure 1. $LP(CQ)$ programs are interpreted as linear programs whose variables describe the solutions of conjunctive queries. As a consequence, they do *not* contain any explicit linear program variables. Instead, they may contain weight expressions $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ over conjunctive queries $Q, Q' \in C_{\mathrm{Q}\Sigma}$. Intuitively, this expression is interpreted as a linear expression over linear program variables representing a solution of $Q \wedge Q'$. Variables of Q and Q' however may be bound in the context, for example through universal quantifiers or Σ-operators. The query variables in $\mathbf{x}$ are bound by the expression taking scope over Q and Q'. The free (query) variable of weight expressions must however be bound by the context, so that they will be instantiated to some database values before evaluation. Weight expressions without free variables reason about an unknown weighting of the answer set of query Q on the given database $\mathbb{D}$ with the variables in $set(\mathbf{x})$. Its value is then the sum over the weights of tuples in the answer set of $Q \wedge Q'$ on the database $\mathbb{D}$ with variables in $set(\mathbf{x})$.

Beside weight expressions, $LP(CQ)$ expressions in $L_{\mathrm{E}\Sigma}$ may also contain expression $N \in Num_{\mathcal{C}}$ or NS where $S \in L_{\mathrm{E}\Sigma}$ and N is a constant number expression, which is either a real number $c \in \mathbb{R}$ or a number expression $\mathbf{num}(E)$ with $E \in \mathcal{X} \cup \mathcal{C}$. An expression $\mathbf{num}(a)$ denotes the real number $\mathbf{num}^{\mathbb{D}}(a^{\mathbb{D}})$ if this value is defined. Note that the real value of $\mathbf{num}(a)$ over $\mathbb{D}$ is constant from the perspective of the linear program once the database $\mathbb{D}$ is fixed. $LP(CQ)$ constrains $C \in L_{\mathrm{C}\Sigma}$ are conjunctions of inequalities $S \leq S'$ between $LP(CQ)$ expressions $S, S' \in L_{\mathrm{E}\Sigma}$, and universally quantified constraints $\forall \mathbf{x}{:}Q.C'$ requiring that C' must be valid for all possible values of $\mathbf{x}$ in the solution of Q over the database (after instantiation of the free variables of $\forall \mathbf{x}{:}Q.C'$). The bound variables in $\mathbf{x}$ have scope over Q and C.

$$
\begin{array}{ll}
fv(c) = \emptyset & fv(\mathbf{num}(E)) = fv(E) \\
fv(\mathbf{weight}_{\mathbf{x}:Q'}(Q)) = fv(Q) \cup fv(Q') \setminus set(\mathbf{x}) & fv(\sum_{\mathbf{x}:Q} S) = fv(S) \cup fv(Q) \setminus set(\mathbf{x}) \\
fv(NS) = fv(N) \cup fv(S) & fv(S \leq S') = fv(S) \cup fv(S') \\
fv(S + S') = fv(S) \cup fv(S') & fv(C \wedge C') = fv(C) \cup fv(C') \\
fv(\forall \mathbf{x}{:}Q.\ C) = fv(Q) \cup fv(C) \setminus \{\mathbf{x}\} & fv(true) = \emptyset \\
fv(\mathbf{maximize}\ S\ \mathbf{subject\ to}\ C) = \emptyset &
\end{array}
$$

Figure 5 Free variables of $LP(CQ)$ expressions, constraints, and programs.

$LP(CQ)$ programs $L \in L\mathrm{P}_\Sigma$ are built from $LP(CQ)$ expressions and $LP(CQ)$ constraints as one might expect. Note, however, that free query variables are ruled out at this level, while being permitted in nested $LP(CQ)$ constraints in $L\mathrm{C}_\Sigma$ and expressions in $L\mathrm{E}_\Sigma$.

The sets of free variables of $LP(CQ)$ expressions, constraints, and programs are formally defined in Figure 5. For instance, the following $LP(CQ)$ constraint C from the warehouse example has three free variables in $fv(C) = \{f, o, q\}$:

$$\mathbf{weight}_{(f',w',b',o'):f' \doteq f \wedge o' \doteq o}(dlr(f', w', b', o')) \leq \mathbf{num}(q)$$

The variables f', w', b', o' are bound by the weight expression. The free variables f, o, q are bound by a quantifier in the context, which in the resource delivery example is the universal quantifier $\forall(f, o, q){:}prod(f, o, q)$.

3.2 Semantics

We next define the semantics of an $LP(CQ)$ program $L \in L\mathrm{P}_\Sigma$ with respect to a database $\mathbb{D} \in db_\Sigma$ with real numbers by an interpretation to a linear program $\langle L \rangle^{\mathbb{D}} \in L\mathrm{P}$, that we will refer to as the naive interpretation from now on.

For doing so, one step is to replace the free variables of the queries of $LP(CQ)$ programs by elements from the database. For this we assume that we have constants for all elements of the database domain, that is $dom(\mathbb{D}) \subseteq \mathcal{C}$. We then define for any conjunctive query Q and variable assignment $\gamma : Y \to D$ a conjunctive query $sbs_\gamma(Q)$, by replacing in Q all free occurrences of variables $y \in Y$ in Q by $\gamma(y)$. The formal definition is given in the full version [2].

In order to define the semantics of an $LP(CQ)$ program L over a database $\mathbb{D}$ we consider the following set of linear program variables:

$$\Theta_L^{\mathbb{D}} = \{\theta^{\alpha}_{sbs_\gamma(Q)} \mid S = \mathbf{weight}_{\mathbf{x}:Q'}(Q) \text{ in } L, \alpha : set(\mathbf{x}) \to dom(\mathbb{D}),\ \gamma : fv(S) \to dom(\mathbb{D})\}$$

Let $S = \mathbf{weight}_{\mathbf{x}:Q'}(Q)$ be a weight expression and $\gamma : Y \to dom(\mathbb{D})$ a variable assignment for the free variables $fv(S) \subseteq Y$ such that $set(\mathbf{x}) \cap Y = \emptyset$. The interpretation of the weight expression $\langle S \rangle^{\mathbb{D},\gamma}$ is the overall weight of the solutions $\alpha \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q' \wedge Q))$ where $\tilde{\gamma} = \gamma_{|Y \setminus set(\mathbf{x})}$ in the table $sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q))$. It is described by the following linear expression:

$$\langle S \rangle^{\mathbb{D},\gamma} = \sum_{\alpha \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q \wedge Q'))} \theta^{\alpha}_{sbs_{\tilde{\gamma}}(Q)}$$

The (naive) interpretations $\langle S \rangle^{\mathbb{D},\gamma}$ and $\langle C \rangle^{\mathbb{D},\gamma}$ of other kinds of $LP(CQ)$ expressions $S \in L\mathrm{E}_\Sigma$ and constraints $C \in L\mathrm{C}_\Sigma$ over a database $\mathbb{D}$ and an environment γ are rather obvious. Note that $LP(CQ)$ programs L can be interpreted as linear programs $\langle L \rangle^{\mathbb{D}} \in L\mathrm{P}$ without any environment as they do not have free variables. The definitions are summarized in Figure 6.

$$\langle \mathbf{weight}_{\mathbf{x}:Q'}(Q)\rangle^{\mathbb{D},\gamma} = \sum_{\alpha \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q \wedge Q'))} \theta^{\alpha}_{sbs_{\tilde{\gamma}}(Q)}$$
$$\langle \forall \mathbf{x}{:}Q.C\rangle^{\mathbb{D},\gamma} = \bigwedge_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q))} \langle C\rangle^{\mathbb{D},\tilde{\gamma}\cup\gamma'}$$
$$\langle \textstyle\sum_{\mathbf{x}:Q} S\rangle^{\mathbb{D},\gamma} = \sum_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q))} \langle S\rangle^{\mathbb{D},\tilde{\gamma}\cup\gamma'}$$
$$\langle NS\rangle^{\mathbb{D},\gamma} = \langle N\rangle^{\mathbb{D},\gamma}\langle S\rangle^{\mathbb{D},\gamma}$$
$$\langle \mathbf{num}(a)\rangle^{\mathbb{D},\gamma} = \mathbf{num}^{\mathbb{D}}(a^{\mathbb{D}}) \qquad \text{(may be undefined)}$$

$$\langle S_1 + S_2\rangle^{\mathbb{D},\gamma} = \langle S_1\rangle^{\mathbb{D},\gamma} + \langle S_2\rangle^{\mathbb{D},\gamma}$$
$$\langle S_1 \leq S_2\rangle^{\mathbb{D},\gamma} = \langle S_1\rangle^{\mathbb{D},\gamma} \leq \langle S_2\rangle^{\mathbb{D},\gamma}$$
$$\langle C_1 \wedge C_2\rangle^{\mathbb{D},\gamma} = \langle C_1\rangle^{\mathbb{D},\gamma} \wedge \langle C_2\rangle^{\mathbb{D},\gamma}$$
$$\langle true\rangle^{\mathbb{D},\gamma} = true$$
$$\langle c\rangle^{\mathbb{D},\gamma} = c$$

$$\langle \mathbf{maximize}\ S\ \mathbf{subject\ to}\ C\rangle^{\mathbb{D}} = \mathbf{maximize}\ \langle S\rangle^{\mathbb{D},\emptyset}\ \mathbf{subject\ to}\ \langle C\rangle^{\mathbb{D},\emptyset}$$

Figure 6 Naïve interpretation of linear expressions (constraints and programs) with conjunctive queries F over database $\mathbb{D}$ as standard linear expression (constraints and programs respectively) $F^{\mathbb{D},\gamma}$, where $\gamma : Y \to dom(\mathbb{D})$ and $fv(F) \subseteq Y \subseteq \mathcal{X}$ and $\tilde{\gamma} = \gamma_{|Y \setminus set(\mathbf{x})}$.

We note that α-renaming the bound variables in weight expressions does *not* always preserve the semantics of $LP(CQ)$ programs. It may make previously equal queries different, so that different weights may be assigned to their answer sets.

3.3 Simple example

We start with the conjunctive query $Q = R(x') \wedge R(y')$ and the database $\mathbb{D}$ with table $R^{\mathbb{D}} = \{(0),(1)\}$. The answer set of Q is $sol^{\mathbb{D}}(Q) = \{\alpha \mid \alpha : \{x', y'\} \to \{0,1\}\}$. We then consider the following $LP(CQ)$ program L:

$$\begin{array}{ll} \mathbf{maximize} & \mathbf{weight}_{(x',y'):true}(Q) \\ \mathbf{subject\ to} & \forall(x){:}R(x).\mathbf{weight}_{(x',y'):x' \doteq x}(Q) \leq 1 \end{array}$$

The naive interpretation $\langle L\rangle^{\mathbb{D}}$ is the following linear program with variables in $\Theta^{\mathbb{D}}_{L}$, where we denote any query answer $\alpha \in sol^{\mathbb{D}}(Q)$ by a pair $(\alpha(x'), \alpha(y'))$ in the cartesian product $\{0,1\}^2$ for simplicity:

$$\begin{array}{rl} \mathbf{maximize} & \theta_Q^{(0,0)} + \theta_Q^{(0,1)} + \theta_Q^{(1,0)} + \theta_Q^{(1,1)} \\ \mathbf{subject\ to} & \theta_Q^{(0,0)} + \theta_Q^{(0,1)} \leq 1 \\ \wedge & \theta_Q^{(1,0)} + \theta_Q^{(1,1)} \leq 1 \end{array}$$

The objective function $\theta_Q^{(0,0)} + \theta_Q^{(0,1)} + \theta_Q^{(1,0)} + \theta_Q^{(1,1)}$ is the naive interpretation of the expression $\mathbf{weight}_{(x',y'):true}(Q)$. The left-hand side of the first constraint $\theta_Q^{(0,0)} + \theta_Q^{(0,1)}$ is obtained by naivly interpreting the expression $\mathbf{weight}_{(x',y'):x' \doteq x}(Q)$ with free variable x in the environment $[x/0]$. Note that these sums share the linear program variables $\theta_Q^{(0,0)}$ and $\theta_Q^{(0,1)}$, so the two weight expressions of L are semantically related.

3.4 Hardness of solving $LP(CQ)$ programs

In this section, we consider study the complexity of the problem $\mathsf{DECIDE}_{\neq 0}(LP(CQ))$ of deciding whether the optimal value of $\langle L\rangle^{\mathbb{D}}$ given an $LP(CQ)$ L and a database $\mathbb{D}$ is non-zero.

▶ **Theorem 6.** $\mathsf{DECIDE}_{\neq 0}(LP(CQ))$ *is NP-hard.*

Proof. The proof follows by reduction to the problem of deciding whether $sol^{\mathbb{D}}(Q) \neq \emptyset$ given a conjunctive query Q and a database $\mathbb{D}$ in the input, which is known to be *NP*-complete [3]. We consider, given a conjunctive query Q, the following $LP(CQ)$: $L_Q :=$ **maximize weight**$_{\mathbf{x}:true}(Q)$ **subject to weight**$_{\mathbf{x}:true}(Q) \leq 1$.

The hardness now follows from the fact that for every Q and $\mathbb{D}$, $\langle L_Q \rangle^{\mathbb{D}} \neq 0$ if and only if $sol^{\mathbb{D}}(Q) \neq \emptyset$. Indeed, if $sol^{\mathbb{D}}(Q) = \emptyset$, then $\langle L_Q \rangle^{\mathbb{D}} =$ **maximize** 0 **subject to** $0 \leq 1$. The optimal value of $\langle L_Q \rangle^{\mathbb{D}}$ is thus 0. However, if $sol^{\mathbb{D}}(Q) \neq \emptyset$, we have $\langle L_Q \rangle^{\mathbb{D}} =$ **maximize** ϕ **subject to** $\phi \leq 1$ where $\phi = \sum_{\alpha \in sol^{\mathbb{D}}(Q)} \theta^{\alpha}_{Q}$. Let $\alpha \in sol^{\mathbb{D}}(Q)$ and consider the weighting ω_α such that $\omega_\alpha(\theta^{\alpha}_{Q}) := 1$ and for every $\alpha' \in sol^{\mathbb{D}}(Q)$ such that $\alpha' \neq \alpha$, $\omega_\alpha(\theta^{\alpha'}_{Q}) := 0$. This weighting clearly respects the constraints of $\langle L_Q \rangle^{\mathbb{D}}$ and has value 1, showing that the optimal value of $\langle L_Q \rangle^{\mathbb{D}} \geq 1$. ◀

4 An Efficiently Solvable Fragment

Motivated by Theorem 6, we now look for a tractable fragment of $LP(CQ)$. We introduce a subclass of projecting $LP(CQ)$ programs and define a notion of width of $LP(CQ)$ programs in this fragment through a collection of hypertree decompositions of the queries they contain. We then show one can find the optimal solution of such programs L more efficiently than by explicitly computing the interpretation over a database $\mathbb{D}$ as a linear program $\langle L \rangle^{\mathbb{D}}$. For this we will present an alternative factorized interpretation of L to a linear program having fewer variables, while preserving the optimal solution.

4.1 Projecting $LP(CQ)$ Programs

We start with the definition of projecting $LP(CQ)$ programs, whose main restriction resides on how they can use conjunctive queries.

▶ **Definition 7.** *The fragment $LP(CQ)_{proj}$ is the set of $LP(CQ)$ programs L such that:*

- *for any subexpression $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ of L, we have that $set(\mathbf{x}) = fv(Q)$ and Q' is a projecting query of the form $\mathbf{x}' \doteq \mathbf{y}$ with $set(\mathbf{x}') \subseteq set(\mathbf{x})$ and $set(\mathbf{x}) \cap set(\mathbf{y}) = \emptyset$.*
- *for any sum $\sum_{\mathbf{x}:Q} S$ and any universal quantifier $\forall \mathbf{x}{:}Q.C$ of L, the query Q is of the form $\exists \mathbf{z}.r(\mathbf{y})$ for some relation symbol $r \in \mathcal{R}^{(n)}$, vector $\mathbf{y} \in \mathcal{X}^n$ and vector $\mathbf{z} \in \mathcal{X}^*$ such that $set(\mathbf{x}) \subseteq fv(Q)$.*

We denote by $LP(CQ_{qf})_{proj}$ the subset of $LP(CQ)_{proj}$ where every conjunctive query Q appearing in a weight expression is quantifier free.

Any expression $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ of a projecting $LP(CQ)$ program is restricted to projection in Q'. Furthermore Q may not have any variables that are free in the weight expression. This condition ensures that the interpretation in environment γ of Q does not substitute any variables, that is $sbs_\gamma(Q) = Q$. Thus, it is interpreted as a sum over θ^{α}_{Q} variables where α are solutions of Q taking the same value $\gamma(\mathbf{y})$ on variables $\mathbf{x}'$. Our algorithm will exploit this fact by utilizing tree decompositions of Q to interpret $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ of $LP(CQ_{qf})_{proj}$ with one variable instead of $|sol^{\mathbb{D}}(Q \wedge Q')|$ needed in the naive interpretation.

Another restriction of $LP(CQ)_{proj}$ is that universal quantifiers and sums are guarded by a database relation. Our algorithm does not exploit the structure of conjunctive queries in universal quantifiers and sums so we interpret these expressions in the same way as in Figure 6. To avoid a blow up in the number of constraints, we chose to guard these constructions.

Hypertree Width of Projecting $LP(CQ)$ Programs. We next lift the concept of generalized hypertree width from quantifier free conjunctive queries to $LP(CQ_{qf})_{proj}$ programs. The complexity of our algorithm will depend on this.

For any program L in $LP(CQ)_{proj}$, we define the set of queries $cqs(L)$ that are weighted when interpreting L as $cqs(L) = \{Q \mid \mathbf{weight}_{\mathbf{x}:Q'}(Q)$ is a subexpression of $L\}$. Observe that the resource delivery problem L is in $LP(CQ)_{proj}$ with $cqs(L) = \{dlr(f', w', b', o')\}$.

▶ **Definition 8.** *Let L be an $LP(CQ_{qf})_{proj}$ program and $\mathcal{T} = (T_Q)_{Q \in cqs(L)}$ a collection of decomposition trees. We call $\mathcal{T}$ a tree decomposition of L if for any expression $\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q)$ in L, $T_Q = (\mathcal{V}_Q, \mathcal{E}_Q, \mathcal{B}_Q)$ is a tree decomposition of Q and there is a node u of T_Q such that $\mathcal{B}_Q(u) = set(\mathbf{x}')$. We define the width of $\mathcal{T}$ to be the maximal width of T_Q for $Q \in cqs(L)$. The size of $\mathcal{T}$ is defined to be $|\mathcal{T}| = \sum_{Q \in cqs(L)} |\mathcal{V}_Q|$.*

The goal of this section is to show how one can take advantage of such tree decompositions in order to solve an $LP(CQ_{qf})_{proj}$ more efficiently than the naive approach of computing its interpretation. To do so, we will use the tree decomposition to compute a smaller interpretation of the linear program that we call *factorized interpretation* and whose definition is given in Section 4.2. The correctness of our approach relies on the following theorem.

▶ **Theorem 9** (Main). *Let L be a $LP(CQ_{qf})_{proj}$ program, $\mathcal{T}$ a decomposition of L of width k and $\mathbb{D}$ a database. The factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ from Figure 7 has $O(|\mathcal{T}||\mathbb{D}|^k)$ variables and the same optimal value as the naïve interpretation $\langle L \rangle^{\mathbb{D}}$.*

Observe that the number of variables of $\langle L \rangle^{\mathbb{D}}$ is roughly the total number of solutions of the conjunctive queries in $cqs(L)$, which may be up to $O(|\mathbb{D}|^n)$, where n is the number of atoms in the conjunctive queries. In Theorem 9, the degree of the polynomial now only depends on the width of the queries, which may be much smaller, resulting in a more succinct linear program that is easier to solve. In the resource optimization example, this allows to go from a cubic number of variables to a quadratic one, but the improvement may be much better on queries having many atoms and small width.

4.2 Factorized Interpretation

The idea of the factorized interpretation is to reduce the number of variables by introducing "group" variables. In the simple example from Section 3.3, we can observe that by grouping together the variables $\theta_Q^{(0,0)}$ and $\theta_Q^{(0,1)}$ into a group variable $\xi_{Q,[x/0]}$ and similarly $\theta_Q^{(1,0)}$ and $\theta_Q^{(1,1)}$ into a group variable $\xi_{Q,[x/1]}$ we can reduce the number of variables from four to two, leading to the following linear program:

$$\begin{array}{rl} \textbf{maximize} & \xi_{Q,[x/0]} + \xi_{Q,[x/1]} \\ \textbf{subject to} & \xi_{Q,[x/0]} \leq 1 \\ \wedge & \xi_{Q,[x/1]} \leq 1 \end{array}$$

Using hypertree decompositions we can develop the grouping idea systematically. We start with a projecting $LP(CQ)$ program L with quantifier free conjunctive queries. Let $\mathcal{T} = (T_Q)_{Q \in cqs(L)}$ be a tree decomposition of L of width k where $T_Q = (\mathcal{V}_Q, \mathcal{E}_Q, \mathcal{B}_Q)$. The set of group variables for the factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ is defined by:

$$\Xi_L^{\mathcal{T},\mathbb{D}} = \{\xi_{Q,u,\beta} \mid Q \in cqs(L), u \in \mathcal{V}_Q, \beta \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)}\}.$$

Observe that since T_Q is a tree decomposition of Q of width at most k, $sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)}$ is of size at most $|\mathbb{D}|^k$. Thus we have at most $|\mathcal{T}||\mathbb{D}|^k$ group variables in $\Xi_L^{\mathcal{T},\mathbb{D}}$.

The *$\mathcal{T}$-factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ of L* is defined in Figure 7. Since we are using the group variables from $\Xi_L^{\mathcal{T},\mathbb{D}}$, it uses at most $|\mathcal{T}||\mathbb{D}|^k$ many variables as stated in Theorem 9. It mainly mirrors the naive interpretation of Figure 6 but significantly differs in two places: the first one is the way $\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q)$ is interpreted and the second one is the addition of the *local soundness constraints* $lsc^{\mathcal{T},\mathbb{D}}(Q)$ to the program.

$$\rho^{\mathcal{T},\mathbb{D},\gamma}(\forall \mathbf{x}{:}r(\mathbf{x}).C) = \bigwedge_{\gamma' \in sol^{\mathbb{D}}(r(\mathbf{x}))} \rho^{\mathcal{T},\mathbb{D},\gamma\cup\gamma'}(C) \qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(S_1 + S_2) = \rho^{\mathcal{T},\mathbb{D},\gamma}(S_1) + \rho^{\mathcal{T},\mathbb{D},\gamma}(S_2)$$
$$\rho^{\mathcal{T},\mathbb{D},\gamma}(\sum_{\mathbf{x}:r(\mathbf{x})} S) = \sum_{\gamma' \in sol^{\mathbb{D}}(r(\mathbf{x}))} \rho^{\mathcal{T},\mathbb{D},\gamma\cup\gamma'}(S) \qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(S_1 \leq S_2) = \rho^{\mathcal{T},\mathbb{D},\gamma}(S_1) \leq \rho^{\mathcal{T},\mathbb{D},\gamma}(S_2)$$
$$\rho^{\mathcal{T},\mathbb{D},\gamma}(NS) = \rho^{\mathcal{T},\mathbb{D},\gamma}(N)\rho^{\mathcal{T},\mathbb{D},\gamma}(S) \qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(C_1 \wedge C_2) = \rho^{\mathcal{T},\mathbb{D},\gamma}(C_1) \wedge \rho^{\mathcal{T},\mathbb{D},\gamma}(C_2)$$
$$\rho^{\mathcal{T},\mathbb{D},\gamma}(\mathbf{num}(a)) = \mathbf{num}^{\mathbb{D}}(a^{\mathbb{D}}) \qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(true) = true$$
(may be undefined) $\qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(c) = c$

$$\rho^{\mathcal{T},\mathbb{D},\gamma}(\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)) = \begin{cases} \xi_{Q,u,\beta} & \text{if } \beta = [\mathbf{x}'/\gamma(\mathbf{y})] \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)} \\ 0 & \text{else} \end{cases}$$

where u is a node of T_Q such that $set(\mathbf{x}') = \mathcal{B}_Q(u)$.

$$\rho^{\mathcal{T},\mathbb{D}}(\mathbf{maximize}\ S\ \mathbf{subject\ to}\ C) = \mathbf{maximize}\ \rho^{\mathcal{T},\mathbb{D},\emptyset}(S)\ \mathbf{subject\ to}\ \rho^{\mathcal{T},\mathbb{D},\emptyset}(C) \wedge \bigwedge_{Q\in cqs(L)} lsc^{\mathcal{T},\mathbb{D}}(Q)$$

Local soundness constraints for $Q \in cqs(L)$ where $A = sol^{\mathbb{D}}(Q)$:

$$lsc^{\mathcal{T},\mathbb{D}}(Q) = \bigwedge_{(u,v)\in\mathcal{E}_Q} \bigwedge_{\gamma\in A_{|\mathcal{B}_Q(u)\cap\mathcal{B}_Q(v)}} \sum_{\beta\in A_{|\mathcal{B}_Q(u)},\beta_{|\mathcal{B}_Q(u)}=\gamma} \xi_{Q,u,\beta} \doteq \sum_{\beta'\in A_{|\mathcal{B}_Q(v)},\beta'_{|\mathcal{B}_Q(v)}=\gamma} \xi_{Q,v,\beta'}$$

Figure 7 $\mathcal{T}$-factorized interpretation of $LP(CQ_{qf})_{proj}$ programs with respect to database $\mathbb{D}$.

In the simple example from Section 3.3 we consider the generalized hypertree decomposition $(\mathcal{V},\mathcal{E},\mathcal{B})$ of Q with $\mathcal{V} = \{r,u,v\}$, $\mathcal{E} = \{(r,u),(r,v)\}$, and $\mathcal{B} = [r/\{\}, u/\{x\}, v/\{y\}]$. The factorized interpretation then yields the following linear program, whose form is similar to the one we showed at the beginning of the subsection:

$$\begin{array}{lll} \mathbf{maximize} & \xi_{Q,r,[]} & \\ \mathbf{subject\ to} & \xi_{Q,u,[x/0]} \leq 1 & \\ \wedge & \xi_{Q,u,[x/1]} \leq 1 & \\ \wedge & \xi_{Q,r,[]} = \xi_{Q,u,[x/0]} + \xi_{Q,u,[x/1]} & \text{local soundness constraints } lsc^{\mathcal{T},\mathbb{D}}(Q) \\ \wedge & \xi_{Q,r,[]} = \xi_{Q,v,[y/0]} + \xi_{Q,v,[y/1]} & \end{array}$$

4.3 Complexity

In this section, we analyse the complexity of constructing the factorized interpretation by applying the inductive definition of Figure 7. Let L be an $LP(CQ)_{proj}$ program and $\mathcal{T} = (T_Q)_{Q\in cqs(L)}$ be a tree decomposition of L of width k where $T_Q = (\mathcal{V}_Q,\mathcal{E}_Q,\mathcal{B}_Q)$.. We first start by bounding the number of variables in $\rho^{\mathcal{T},\mathbb{D}}(L)$. The set of variables of $\rho^{\mathcal{T},\mathbb{D}}(L)$ is, by definition, $\Xi_L^{\mathcal{T},\mathbb{D}}$. Variables are indexed by a query Q of $cqs(L)$, a node of u in the tree decomposition T_Q of Q and a partial assignment β of $\mathcal{B}(Q)$, projected on a bag of T_Q. By Lemma 3, we have at most $|\mathbb{D}|^k$ elements in $sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)}$. Thus, $\Xi_L^{\mathcal{T},\mathbb{D}}$ has size at most $|\mathcal{T}||\mathbb{D}|^k$ and this set of variables can be computed in time $O(|\mathcal{T}||\mathbb{D}|^k \log(|\mathbb{D}|))$ by Lemma 3.

We now evaluate the number of constraints in $\rho^{\mathcal{T},\mathbb{D}}(L)$. It is clear from Figure 7 that $\rho^{\mathcal{T},\mathbb{D}}(L)$ contains the constraints from L where $\forall$ quantifiers have been unfolded plus the local soundness constraints. By definition, there are at most t $|\mathcal{T}||\mathbb{D}|^k$ local soundness constraints. Moreover, once the projection of $sol^{\mathbb{D}}(Q)$ on the bags of T_Q have been precomputed thanks to Lemma 3, it is easy to see that each local soundness constraints can be computed in time $O(|\mathcal{T}||\mathbb{D}|^k \log(|\mathbb{D}|))$ by taking the join of every bags that are neighbor in one tree decomposition.

Now, let $d_\forall$ the the maximum number of nested $\forall$ appearing in L. It is easy to see that $\rho^{\mathcal{T},\mathbb{D}}(\forall \mathbf{x}{:}r(\mathbf{x}).C)$ multiplies the number of constraints in $\rho^{\mathcal{T},\mathbb{D}}(C)$ by $|r| \leq |\mathbb{D}|$. Thus, apart from the local soundness constraints, there are at most $|L||\mathbb{D}|^{d_\forall}$ constraints in $\rho^{\mathcal{T},\mathbb{D}}(L)$.

Finally, we turn to the complexity of constructing these constraints of $\rho^{\mathcal{T},\mathbb{D}}(L)$. To evaluate the constraints of $\rho^{\mathcal{T},\mathbb{D}}(L)$, one has to unfold $\sum_{\mathbf{x}:r(\mathbf{x})} S$ statements. Let d_Σ be the maximum number of nested $\sum$ in L. Each unfolding of a $\sum_{\mathbf{x}:r(\mathbf{x})} S$ leads to at most $|r| \leq |\mathbb{D}|$ inductive construction. Thus, in a quantifier free constraint from L, one has to evaluate at most $|\mathbb{D}|^{d_\Sigma}$ S-terms. The main complexity of evaluating such term comes from computing $\rho^{\mathcal{T},\mathbb{D}}(\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q))$. Each of them can easily be computed by scanning the structure given by Lemma 3 in time $O(|\mathcal{T}||\mathbb{D}|^k)$.

Overall, the complexity of constructing $\rho^{\mathcal{T},\mathbb{D}}(L)$ is thus $O(|\mathcal{T}||\mathbb{D}|^k(\log|\mathbb{D}|+|\mathbb{D}|^k+|\mathbb{D}|^{d_\forall+d_\Sigma})$.

4.4 Correctness

We now discuss how to prove the correctness statement of Theorem 9 that $\rho^{\mathcal{T},\mathbb{D}}(L)$ has the same optimal value as $\langle L\rangle^{\mathbb{D}}$.

One can see that given a context γ such that $\rho^{\mathcal{T},\mathbb{D},\gamma}(\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)) = \xi_{Q,u,\beta}$, the usual interpretation would have been $\langle\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)\rangle^{\mathbb{D},\gamma} = \sum_{\alpha\in sol^{\mathbb{D}}_{set(\mathbf{x})}(Q):\alpha|_{\mathbf{x}'}=\beta}\theta^\alpha_Q$, that is, intuitively, $\xi_{Q,u,\beta}$ represents the linear sum of variables θ^α_Q in the naive interpretation with α compatible with β.

For general programs L we will reconstruct a solution to $\langle L\rangle^{\mathbb{D}}$ from a solution to $\rho^{\mathcal{T},\mathbb{D}}(L)$ such that the value of $\xi_{Q,u,\beta}$ indeed corresponds to the sum of the values of variables θ^α_Q with α compatible with β and vice-versa. To ensure that this is always possible, we have to be careful that variables $\xi_{Q,u,\beta}$ and $\xi_{Q,v,\beta'}$ are compatible with one another because they may correspond to two sums on θ^α_Q variables having a non-empty intersection. We ensure this through *local soundness constraints* $lsc^{\mathcal{T},\mathbb{D}}(Q)$ for every query $Q \in cqs(L)$.

Weightings on Tree Decompositions. One can observe that the key idea in the definition of $\rho^{\mathcal{T},\mathbb{D}}(L)$ is to introduce linear program variables that will intuitively encode the sum of several linear program variables in the naive interpretation $\langle L\rangle^{\mathbb{D}}$. A solution to $\langle L\rangle^{\mathbb{D}}$ maps a variable θ^α_Q to a non-negative real number where $\alpha \in sol^{\mathbb{D}}(Q)$. In other words, it assigns a weight $\omega(\alpha) \in \mathbb{R}_+$ to every $\alpha \in sol^{\mathbb{D}}(Q)$ for every $Q \in cqs(L)$. A solution to $\rho^{\mathcal{T},\mathbb{D}}(L)$ maps a variable $\xi_{Q,u,\beta}$ to a non-negative real number where $\beta \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)}$. In other words, it assigns a weight W_u to every β that is in $sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ for every node u of T_Q.

To reconstruct a solution of $\langle L\rangle^{\mathbb{D}}$ from a solution W of $\rho^{\mathcal{T},\mathbb{D}}(L)$, we need to be able to reconstruct a weighting ω of $sol^{\mathbb{D}}(Q)$ such that $\sum_{\alpha|_{\mathcal{B}_Q(u)}=\beta}\omega(\alpha) = W_u(\beta)$. In this section, we explain that this is always possible as long as the W_u are compatible with one another, which is ensured by local soundness constraints $lsc^{\mathcal{T},\mathbb{D}}(Q)$ in $\rho^{\mathcal{T},\mathbb{D}}(L)$.

The technique is not specifically tied to the fact that the weights are assigned to the solutions of a quantifier free conjunctive query, thus we formulate our result in a more general setting by considering weightings on a set $A \subseteq D^X = \{\alpha \mid \alpha : X \to D\}$ for a finite set of variables X. Intuitively however, one can think of A as $sol^{\mathbb{D}}(Q)$ for **a quantifier-free conjunctive query** Q.

We start by introducing a few notations. Let $X' \subseteq X \subseteq \mathcal{X}$. For any $\alpha' : X' \to D$ we define the set of its extensions into A by $A[\alpha'] = \{\alpha \in A \mid \alpha_{|X'} = \alpha'\}$. Moreover, given a weighting $\omega : A \to \mathbb{R}_+$ of A, we define the projection $\pi_{X'}(\omega) : A_{|X'} \to \mathbb{R}_+$ such that for all $\alpha' \in A_{|X'}$: $\pi_{X'}(\omega)(\alpha') = \sum_{\alpha\in A[\alpha']}\omega(\alpha)$.

We now fix $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ a decomposition tree for X. Given two nodes $u, v \in \mathcal{V}$ we denote the intersection of their bags by $\mathcal{B}^{uv} = \mathcal{B}(u) \cap \mathcal{B}(v)$.

▶ **Definition 10.** *A family $W = (W_v)_{v \in \mathcal{V}}$ is a* weighting collection on T for A *if it satisfies the following conditions for any two nodes $u, v \in \mathcal{V}$:*
- **W_u is a weighting of $A_{|\mathcal{B}(u)}$**, *i.e.,* $W_u : A_{|\mathcal{B}(u)} \to \mathbb{R}_+$.
- **W_u is sound for T at $\{u, v\}$**, *i.e.,* $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(W_v)$.

Intuitively, the soundness of a weighting collection on T is a minimal requirement for the existence of a weighting ω of A such that W_u is the projection of ω on the bag $\mathcal{B}(u)$ of T, that is $W_u = \pi_{\mathcal{B}(u)}(\omega)$ since we have the following:

▶ **Proposition 11.** *For any weighting $\omega : A \to \mathbb{R}_+$, the family $(\pi_{\mathcal{B}(v)}(\omega))_{v \in \mathcal{V}}$ is a weighting collection on T for A.*

What is more interesting is the other way around, that is, given $(W_u)_{u \in \mathcal{V}}$ a weighting collection on T, whether we can find a weighting ω of A such that $W_u = \pi_{\mathcal{B}(u)}(\omega)$ for every u. It turns out that soundness is not enough to ensure the existence of such a weighting. However it becomes possible when A is conjunctively decomposed:

▶ **Definition 12.** *Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree of $X \subseteq \mathcal{X}$. We call a subset of variable assignments $A \subseteq D^X$* conjunctively decomposed by T *if for all $u \in \mathcal{V}$ and $\beta \in A_{|\mathcal{B}(u)}$: $\{\alpha_1 \cup \alpha_2 \mid \alpha_1 \in A_{|\mathcal{B}(\uparrow u)}[\beta],\ \alpha_2 \in A_{|\mathcal{B}(\downarrow u)}[\beta]\} \subseteq A[\beta]$ where $\mathcal{B}(V) = \bigcup_{v \in V} \mathcal{B}(v)$ for any $V \subseteq \mathcal{V}$.*

Note that the inverse inclusion holds in general. Of course, this property holds if A is the answer set of a conjunctive queries and the tree is a tree decomposition of Q:

▶ **Proposition 13.** *For any tree decomposition T of a quantifier free conjunctive query $Q \in CQ_\Sigma$ and database $\mathbb{D} \in db_\Sigma$, the answer set $sol^{\mathbb{D}}(Q)$ is conjunctively decomposed by T.*

Proposition 13 does not hold when Q is not quantifier free. It explains why the technique only works for the fragment $LP(CQ_{qf})_{proj}$. We however explain how one can use the same technique on $LP(CQ)_{proj}$ in Section 4.5.

Soundness and conjunctive decomposition are enough to prove this correspondence theorem that allows us to transform solutions of $\rho^{\mathcal{T},\mathbb{D}}(L)$ to solutions of $\langle L \rangle^{\mathbb{D}}$ and vice-versa.

▶ **Theorem 14** (Correspondence)**.** *Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a normalized decomposition tree of $X \subseteq \mathcal{X}$ and $A \subseteq D^X$ be a set of variable assignments that is conjunctively decomposed by T.*
1. *For every weighting ω of A, $(\pi_{\mathcal{B}(u)}(\omega))_{u \in \mathcal{V}}$ is a weighting collection on T for A.*
2. *For any weighting collection W on T for A there exists a weighting ω of A such that $\forall u : W_u = \pi_{\mathcal{B}(u)}(\omega)$.*

While the first item of Theorem 14 follows by Proposition 11 and can be proven by a simple calculation, the second item is harder to prove. We present here one way of constructing ω from $(W_u)_{u \in \mathcal{V}}$. The proof of correctness of this construction can be found in the full version [2].

Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a normalized decomposition tree of X and $W = (W_u)_{u \in \mathcal{V}}$ a weighting collection on T for $A \subseteq D^X$. For any node $u \in \mathcal{V}$, we inductively construct $\omega_u : A_{|\mathcal{B}(\downarrow u)} \to \mathbb{R}_+$.

If u is a leaf of T, we define ω_u such that for all $\alpha \in A_{|\mathcal{B}(\downarrow u)}$, $\omega_u(\alpha) := W_u(\alpha)$.

Now, assume $\omega_{u'}$ is defined for all children u' of u. Let $\alpha \in A_{|\mathcal{B}(\downarrow u)}$ and denote by $\beta = \alpha_{|\mathcal{B}(u)}$. We define $\omega_u(\alpha)$ as follows:

If u is an extend node with a single child v then $\omega_u(\alpha) := \frac{W_u(\beta)}{W_v(\alpha_{|\mathcal{B}(v)})}\omega_v(\alpha_{|\mathcal{B}(\downarrow v)})$ if $W_v(\alpha_{|\mathcal{B}(v)}) > 0$ and $\omega_u(\alpha) := 0$ otherwise.

If u is a project node with a single child v then $\omega_u(\alpha) := \omega_v(\alpha_{|\mathcal{B}(\downarrow v)})$.

If u is a join node with children $v_1, \ldots, v_k$ then $\omega_u(\alpha) := \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha_{|\mathcal{B}(\downarrow v_i)})}{W_u(\beta)^{k-1}}$ if $W_u(\beta) > 0$ and $\omega_u(\alpha) := 0$ otherwise.

Finally, we let ω be ω_r where r is the root of T. The proof that $\forall u : W_u = \pi_{\mathcal{B}(u)}(\omega)$ is done via two inductions. The first one is a bottom-up induction to prove that $W_u = \pi_{\mathcal{B}(u)}(\omega)$ for every node u in the tree decomposition. Then, by top-down induction, one can prove that $\omega_u = \pi_{\mathcal{B}(\downarrow u)}(\omega_r)$. The proof is tedious and mainly relies on calculations and careful analysis on how A is decomposed along T.

Correctness Proof. We are now ready to prove that, given a tree decomposition $\mathcal{T}$ of a $LP(CQ)$ program L of $LP(CQ_{qf})_{proj}$, $\rho^{\mathcal{T},\mathbb{D}}(L)$ and $\langle L \rangle^{\mathbb{D}}$ have the same optimal value.

For any weighting $\dot{\omega}$: $\Theta_L \to \mathbb{R}_+$ we define a weighting $\Pi(\dot{\omega}) : \Xi_L^{\mathcal{T},\mathbb{D}} \to \mathbb{R}_+$ such that for all $\xi_{Q,u,\beta} \in \Xi_L^{\mathcal{T},\mathbb{D}}$: $\Pi(\dot{\omega})(\xi_{Q,u,\beta}) = \sum_{\alpha \in sol^{\mathbb{D}}(Q)[\beta]} \dot{\omega}\ (\theta_Q^{\alpha})$.

Observe that $\dot{\omega}$ can be seen as a collection of weightings of $sol^{\mathbb{D}}(Q)$ for $Q \in cqs(L)$. It turns out that evaluating linear expressions and constraints of $\langle L \rangle^{\mathbb{D}}$ with $\dot{\omega}$ returns the same value as the evaluation of linear expressions and constraints of $\rho^{\mathcal{T},\mathbb{D}}(L)$ with $\Pi(\dot{\omega})$:

▶ **Lemma 15.** *For any $\mathcal{T}-$projecting sum $S \in L_{E_\Sigma}$ and environment $\gamma : X \to dom(\mathbb{D})$ where $fv(S) \subseteq X$ it holds that $[\![\langle S \rangle^{\mathbb{D},\gamma}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(S)]\!]_{\Pi(\dot{\omega})}$.*

▶ **Lemma 16.** *For any constraint $C \in L_{C_\Sigma}$ that is $\mathcal{T}-$projecting and environment $\gamma : X \to dom(\mathbb{D})$ where $fv(C) \subseteq X$: $[\![\langle C \rangle^{\mathbb{D},\gamma}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(C)]\!]_{\Pi(\dot{\omega})}$.*

Lemma 15 and Lemma 16 rely on Proposition 11. It is easy to see that they imply that if $\dot{\omega}$ is a solution of $\langle L \rangle^{\mathbb{D}}$ (the fact that it respects the local soundness constraints follows from Proposition 11), then $\Pi(\dot{\omega})$ is a solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$ with the same value. Thus, the optimal value of $\rho^{\mathcal{T},\mathbb{D}}(L)$ is greater or equal than the optimal value of $\langle L \rangle^{\mathbb{D}}$.

To prove the equality, it remains to prove that the optimal value of $\langle L \rangle^{\mathbb{D}}$ is greater or equal than the optimal value of $\rho^{\mathcal{T},\mathbb{D}}(L)$. To this end, consider a solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$. It is a weighting $\dot{W}$ of $\Xi_L^{\mathcal{T},\mathbb{D}}$ which respects the local soundness constraints. By Theorem 14, we will be able to reconstruct a weighting $\dot{\omega}$ of Θ_L which respects the constraint of $\langle L \rangle^{\mathbb{D}}$. It is formalized in the following lemma whose proof can be found in the the full version [2].

▶ **Lemma 17.** *For any weighting $\dot{W}$ of $\Xi_L^{\mathcal{T},\mathbb{D}}$ such that $[\![\bigwedge_{Q \in \mathcal{Q}} lsc^{\mathcal{T},\mathbb{D}}(Q)]\!]_{\dot{W}} = 1$, there exists a weighting $\dot{\omega}$ of $\Theta_{\mathcal{Q}}$ such that $\dot{W} = \Pi(\dot{\omega})$.*

▶ **Proposition 18.** *Let $\mathbb{D}$ be a database and $\mathcal{T}$ a collection of decomposition tree. Any $\mathcal{T}-$projecting $LP(CQ)$ program $L = (\textbf{\textit{maximize}}\ S\ \textbf{\textit{subject to}}\ C) \in L_{P_\Sigma}$ satisfies that:*

1. *For any solution $\dot{\omega}$ of $\langle L \rangle^{\mathbb{D}}$ there is a solution $\dot{W}$ of $\rho^{\mathcal{T},\mathbb{D}}(L)$ s.t. $[\![\langle S \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D}}(S)]\!]_{\dot{W}}$.*
2. *For any solution $\dot{W}$ of $\rho^{\mathcal{T},\mathbb{D}}(L)$ there is a solution $\dot{\omega}$ of $\langle L \rangle^{\mathbb{D}}$ s.t. $[\![\langle S \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D}}(S)]\!]_{\dot{W}}$.*

4.5 Treatment of Existential Quantifiers

The previous method of factorized interpretation only works for the $LP(CQ_{qf})_{proj}$ fragment, where conjunctive queries are supposed to be quantifier free. It turns out that one can similarly solve linear programs of $LP(CQ)_{proj}$ programs by applying a simple transformation.

For any $LP(CQ)_{proj}$ program L we can move the existential quantifiers of the conjunctive query into the weight expression as follows, yielding an $LP(CQ_{qf})_{proj}$ program $mvq(L)$: we replace every subexpression $\textbf{weight}_{\mathbf{x}:Q'}(\exists \mathbf{z}.Q)$ of L, where Q is quantifier free, by $\textbf{weight}_{\mathbf{xz}:Q'}(Q)$ where $\mathbf{xz}$ is the concatenation of vectors $\mathbf{x}$ and $\mathbf{z}$. We have:

▶ **Theorem 19** (Removing Existential Quantifiers). *For any projecting $LP(CQ)$ program, the $LP(CQ_{qf})_{proj}$ program $mvq(L)$ has the same optimal value as L.*

Observe that we can use this technique for the resource delivery problem L. In $mvq(L)$, there is only one query on variables $(f', o', q, q', b', w', c, c')$. It is easy to see that it has hypertree width 2 since we can construct a tree decomposition having two connected bags $\mathcal{B}(u) = \{f', o', b', q, q'\}$ and $\mathcal{B}(v) = \{f', w', b', c, c'\}$. $\mathcal{B}(u)$ is covered by the first two atoms and $\mathcal{B}(v)$ by the last two. Now, because of the weight expressions, we also need to add a bag for $\{f', w'\}$, $\{w'\}$ and $\{w', b'\}$ which can safely be connected to v, and for $\{f', o'\}$ and $\{b', o'\}$ which can safely be connected to u. It gives a decomposition of L of width 2, showing that the factorized interpretation will have fewer variables than the naive interpretation.

5 Preliminary Experimental Results

The practical performances of our idea heavily depends on how linear solvers perform on factorized interpretation. We compared the performances of GLPK on both the naive interpretation and the factorized interpretation of the resource delivery problem from the introduction using some synthetic data. For each run we fixed an input size m as well as a domain D of size $n = f(m)$. We then genereated each input table of arity k by uniformly sampling m tuples from the n^k possible tuples on D. The value of k was defined so that the ratio of selected tuples $\frac{m}{n^k}$ was constant throughout the runs. We used Python and the Pulp library to build the linear programs as well as a hard-coded tree-decomposition of the *dlr* query. The tests were run on an office laptop by progressively increasing the size of the generated input tables. A summary of our experiments is displayed on Figure 8.

As expected when comparing both linear programs we observed a larger number of constraints (due to the soundness constraints) and a smaller number of variables in the factorized interpretation. While building the naive interpretation quickly became slower than building the factorized interpretation, we did not analyze this aspect further since we are not using a database engine to build the naive interpretation and solve it directly from the tree decomposition, which may not be the fastest method without further optimizations. Most interestingly solving the factorized interpretation was faster than solving the naive interpretation in spite of the increased number of constraints thanks to the decrease in the number of variables. In particular for an instance with an input size of 2000 lines per table, the naive interpretation had roughly 1.5 million variables while the factorized interpretation had only roughly 150000. The solving time was also noticeably improved at 22s for the factorized case against 106s for the naive one.

6 Conclusion and Future Work

Our preliminary experiments seem to confirm the efficiency of factorized interpretation, in accordance with our complexity results. More thorough benchmarking is needed to evaluate the practical relevance though. Another direction to explore would be to better integrate our approach into a database engine, in the way it is done by `SolveDB` for example. Finally, other optimization problems may benefit from this approach such as convex optimization or integer linear programming. It would be interesting to define languages analogous to $LP(CQ)$ for these optimization problems and study how conjunctive query decompositions could help to improve the efficiency.

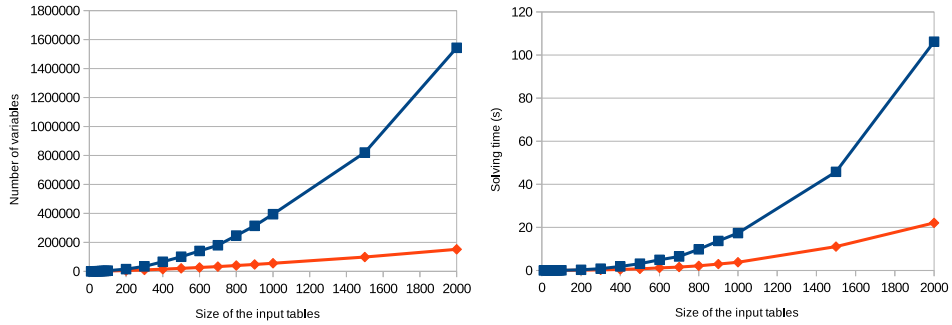


Figure 8 Number of variables and performances of GLPK for naive (blue) and factorized (red) interpretation of the resource delivery problem with respect to table size.

References

1 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.

2 Florent Capelli, Nicolas Crosetti, Joachim Niehren, and Jan Ramon. Linear Programs with Conjunctive Queries. In *25th Internationcal Conference on Database Theory (ICDT 2022)*, Edinburgh, United Kingdom, March 2022. URL: `https://hal.archives-ouvertes.fr/hal-01981553`.

3 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM. `doi:10.1145/800105.803397`.

4 Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

5 Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.

6 G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002. arXiv: cs/9812022.

7 Georg Gottlob, Nicola Leone, and Francesco Scarcello. On tractable queries and constraints. In *International Conference on Database and Expert Systems Applications*, pages 1–15. Springer, 1999.

8 Martin Grohe. The structure of tractable constraint satisfaction problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 58–72. Springer, 2006.

9 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG)*, 11(1):4, 2014.

10 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Comb.*, 4(4):373–396, 1984. `doi:10.1007/BF02579150`.

11 Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

12 Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. Efficient querying of inconsistent databases with binary integer programming. *Proceedings of the VLDB Endowment*, 6(6):397–408, April 2013. `doi:10.14778/2536336.2536341`.

13 Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

14 Alexandra Meliou and Dan Suciu. Tiresias: The database oracle for how-to queries. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 337–348, New York, NY, USA, 2012. ACM. `doi:10.1145/2213836.2213875`.

15 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

16 Dan Olteanu and Jakub Závodnỳ. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory*, pages 285–298. ACM, 2012.

17 Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015.

18 Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79:984–1001, September 2013.

19 Laurynas Šikšnys and Torben Bach Pedersen. SolveDB: Integrating optimization problem solvers into SQL databases. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, page 14. ACM, 2016.

20 Yuyi Wang, Jan Ramon, and Thomas Fannes. An efficiently computable subgraph pattern support measure: counting independent observations. *Data Mining and Knowledge Discovery*, 27(3):444–477, November 2013.

21 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, VLDB '81, pages 82–94. VLDB Endowment, 1981.

Certifiable Robustness for Nearest Neighbor Classifiers

Austen Z. Fan
Department of Computer Sciences, University of Wisconsin-Madison, WI, USA

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

Abstract

ML models are typically trained using large datasets of high quality. However, training datasets often contain inconsistent or incomplete data. To tackle this issue, one solution is to develop algorithms that can check whether a prediction of a model is *certifiably robust*. Given a learning algorithm that produces a classifier and given an example at test time, a classification outcome is certifiably robust if it is predicted by every model trained across all possible worlds (repairs) of the uncertain (inconsistent) dataset. This notion of robustness falls naturally under the framework of certain answers. In this paper, we study the complexity of certifying robustness for a simple but widely deployed classification algorithm, k-Nearest Neighbors (k-NN). Our main focus is on inconsistent datasets when the integrity constraints are functional dependencies (FDs). For this setting, we establish a dichotomy in the complexity of certifying robustness w.r.t. the set of FDs: the problem either admits a polynomial time algorithm, or it is coNP-hard. Additionally, we exhibit a similar dichotomy for the counting version of the problem, where the goal is to count the number of possible worlds that predict a certain label. As a byproduct of our study, we also establish the complexity of a problem related to finding an optimal subset repair that may be of independent interest.

2012 ACM Subject Classification Theory of computation → Database theory; Theory of computation → Incomplete, inconsistent, and uncertain databases

Keywords and phrases Inconsistent databases, k-NN classification, certifiable robustness

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.6

Related Version *Full Version*: `https://arxiv.org/abs/2201.04770` [7]

Funding This research was supported in part by National Science Foundation grants CRII-1850348 and III-1910014, as well as a gift by Google.

1 Introduction

Machine Learning (ML) has been widely adopted as a central tool in business analytics, medical decisions, autonomous driving, and many other domains. In supervised learning settings, ML models are typically trained using large datasets of high quality. However, real-world training datasets often contain incorrect or incomplete data. For example, attribute values may be missing from the dataset, attribute values may be wrong, or the dataset can violate integrity constraints. Several approaches to tackle this problem have been proposed in the literature, including data cleaning [17, 23] and robust learning methods [4, 26, 5].

In this work, we study an alternative but complementary approach using the framework of *certain answers*, which has been a focus of the database community in the last two decades [1, 2]. Under this framework, an inconsistent or incomplete training dataset is modeled as a set of possible worlds called *repairs*. We can think of a repair as a way to clean the dataset such that it is consistent (w.r.t. a set of integrity constraints) and complete. In the context of query answering, certain answers are the output tuples that exist in the query result of every possible world. In other words, we will obtain a certain answer in the output no matter how we choose to repair the dataset.

	A	B	C	label
t_1	1	0	a	0
t_2	1	2	b	0
t_3	2	0	a	2
t_4	2	5	c	1
t_5	3	1	a	0
t_6	4	2	d	2

The notion of certain answers in the context of ML is known as certifiable (or certified) robustness [3, 26]. Given a learning algorithm that produces a classifier and a tuple at test time, we say that a predicted label is *certifiably robust* if it is predicted by every model trained across all possible worlds. In other words, certifiably robust predictions come with a formal guarantee that the label is robust to how the training dataset is fixed. Such a guarantee can be beneficial to decide whether we should trust the predictions of the ML model or whether we should spend resources to clean the dataset before feeding it to the learning algorithm.

As a first step towards certifying robustness for other more complex ML algorithms, we focus in this work on k-Nearest Neighbor classification (k-NN). In this problem, we start with a d-dimensional dataset equipped with a distance function. Given a test point x, the classifier first finds the k points closest to x w.r.t. the given distance function, and assigns to x the label that is a plurality among these. Certified robustness for k-NNs was recently explored by Karlas et. al [13] under the uncertain model where tuples (training points) with the same label are grouped into blocks, and a possible world is constructed by picking independently exactly one tuple from each block. This setting is equivalent to considering the subset repairs of an inconsistent dataset where the only integrity constraint is a primary key, with the additional restriction that training points with the same key must have the same label.

In this paper, we generalize the study of certifying robustness for k-NNs to other inconsistent and uncertain models. Specifically, we consider subset repairs of a dataset where the integrity constraints can be any set of functional dependencies (FDs) and not only a primary key as in [13].

▶ **Example 1.** Consider the inconsistent dataset in the table, where the integrity constraint is the FD $A \to B$. Let the distance function between two tuples s, t be $f(s,t) = |s[A] - t[A]| + |s[B] - t[B]|$. Suppose that we want to label the test point $x = (0, 0, d)$ using a 3-NN classifier. This induces the following ordering of the tuples w.r.t. their distance from x (for convenience, we include the labels as well):

$$t_1 : \mathbf{0} < t_3 : \mathbf{2} < t_2 : \mathbf{0} < t_5 : \mathbf{0} < t_6 : \mathbf{2} < t_4 : \mathbf{1}$$

A repair for this inconsistent dataset has to choose one tuple from $\{t_1, t_2\}$ and one tuple from $\{t_3, t_4\}$. There are 4 possible repairs, which form the following 3-neighborhoods around x:

$$\{t_1 : \mathbf{0}, t_5 : \mathbf{0}, t_6 : \mathbf{2}\}, \qquad \{t_2 : \mathbf{0}, t_5 : \mathbf{0}, t_6 : \mathbf{2}\}$$
$$\{t_1 : \mathbf{2}, t_3 : \mathbf{0}, t_5 : \mathbf{0}\}, \qquad \{t_3 : \mathbf{2}, t_2 : \mathbf{0}, t_5 : \mathbf{0}\}$$

In all repairs, label 0 occurs two times, and hence it is always the majority label. Hence, we can certify that 0 is a robust label for tuple x.

We show that for general sets of FDs the complexity landscape for certified robustness admits a dichotomy: it is computationally intractable for some sets of FDs, and is in polynomial time for the other sets of FDs. We also investigate certifying robustness for

other widely used uncertain models, including Codd-tables [11], or-sets and ?-sets [25]. We establish that in these settings the problem can always be solved in polynomial time.

Our work shows that the logical structure of the errors (or missing values) from a training set can be exploited to construct fast algorithms for certifiable robustness. Tools developed in the database theory community can facilitate the design of these algorithms. We also demonstrate that, even for the relatively simple k-NN classifier, the complexity landscape exhibits a complex behavior that is related to other problems in consistent query answering.

Our Contribution. We now present in more detail the contributions of this work:

- We establish a complexity dichotomy for certifying robustness for k-NNs under subset repairs (Section 4) into P and coNP-complete. The dichotomy depends on the structure of FDs. More precisely, the syntactic condition for the dichotomy is based on the notion of *lhs chains*. In fact, it is the same as the one used for the complexity classification of the problem of counting the number of subset repairs under a set of FDs [21]. In the case where the only FD constraint is a primary key, we show that we can design an even faster algorithm that runs in linear time in the size of the data, improving upon the running time of the algorithm in [13] (Section 5).
- In addition to certified robustness, we study the related problem of *counting* the number of repairs that predict a given label for a test point (Section 8). We establish a dichotomy into the complexity classes FP and #P-complete with the same syntactic condition as the decision problem. The polynomial time algorithm here depends exponentially on the number of classification labels, in contrast to our algorithm for certifiable robustness which has a linear dependence on the number of labels.
- We show that certifying robustness for k-NNs is tightly connected to the problem of finding the *subset repair with the smallest total weight*, when each tuple is assigned a positive weight (Section 7). As a consequence, we obtain a dichotomy result for that problem as well. Note that this problem is a symmetric variant of the problem in [20], which asks instead for the repair with the maximum total weight.
- Finally, we investigate the complexity of certifiable robustness for k-NNs for three widely used uncertain models: Codd-tables, ?-sets and or-sets (Section 9). We show that for all the above models, certifying robustness for k-NN classification admits a polynomial time algorithm.

2 Related Work

Certain Query Answering. There has been a long line of research in *Certain Query Answering* (CQA) in the database community. Data consistency might be violated, for example, during data integration or in a data warehouse. It is then natural to ask: given a query and such inconsistent data, can we answer the query with a certain answer? Arenas, Bertossi, and Chomicki [1] first define the notion of a repair, which refers to a consistent subinstance that is minimally different from the inconsistent data. A certain answer to the query is defined as an answer that will result from every repair. Beginning from the work of Fuxman and Miller [8], more general dichotomy results in CQA have been proven [14, 15, 16]. A dichotomy theorem for a class of queries and integrity constraints says CQA is either in polynomial time or coNP-complete, usually depending on an efficiently checkable criterion for tractability. Certain answers have also been studied in the context of incomplete models of data [19, 22, 10].

Subset Repairs. Livshits et. al [21] studied the problem of counting the number of subset repairs w.r.t. a given set of FDs, establishing a complexity dichotomy. The syntactic condition for tractability (existence of a lhs chain) is the same as the one we establish for certifiable robustness in k-NN classification. Livshits et. al [20] also studied the problem of finding a maximum-weight subset repair w.r.t. a given set of FDs, and showed that the complexity observes a dichotomy. In this paper we study the symmetric problem of finding a minimum-weight subset repair, and show that the problem also exhibits a complexity dichotomy, albeit the condition for tractability is again the existence of a lhs chain.

Certifiable Robustness in ML. Robust learning methods are used to learn over inconsistent or incomplete datasets. For example, [5] discusses a sound verification technique which proves whether a prediction is robust to data poisoning in decision-tree models. There is also a line of work on smoothing techniques for ML robustness [3, 12, 24, 18], where added random noise, usually with a Gaussian distribution, will sometimes boost the robustness of the model against adversarial attacks such as label-flipping. Our approach is different in that we prove a dichotomy for k-NN certifiable robustness, i.e. *either* we can assert that the dataset will always lead to the same prediction efficiently *or* it is coNP-complete to do so, with an efficiently testable criterion. We show how to extend our model to capture scenarios including uncertain labels, weighted tuples, and data poisoning.

3 Preliminaries

In this paper, we consider relation schemas of the form $R(A_1, \ldots, A_d)$ with arity d. The attributes $A_1, \ldots, A_d$ take values from a (possibly infinite) domain $\mathbb{D}$. Given a tuple t in an instance I over R, we will use $t[A_i]$ to denote its value at attribute A_i. It will be convenient to interpret an instance I as a training set of points in the d-dimensional space $\mathbb{D}^d$. We will use the terms point/tuple interchangeably in the paper.

An *uncertain instance* $\mathcal{I}$ over a schema $R(A_1, \ldots, A_d)$ is a set of instances over the schema. We will often refer to each instance in $\mathcal{I}$ as a *possible world*. We will see later different ways in which we can define uncertain instances implicitly.

For each tuple t that occurs in some possible world in $\mathcal{I}$, we associate a *label* $L(t)$ which takes values from a finite set $\mathcal{Y}$. We will say that the uncertain instance $\mathcal{I}$ equipped with the labeling function L is a *labeled uncertain instance* over the schema $R(A_1, \ldots, A_d)$. We similarly define a labeled instance I.

Certifiable Robustness. In this work, we will focus on the classification task. Let $\mathcal{L}$ be a learning algorithm that takes as training set a labeled instance I over the schema $R(A_1, \ldots, A_d)$, and returns a *classifier*, which is a total function $\mathcal{L}_I : \mathbb{D}^d \to \mathcal{Y}$.

▶ **Definition 2** (Certifiable Robustness). *Let $\mathcal{I}$ be a labeled uncertain instance over $R(A_1, \ldots, A_d)$ and $\mathcal{L}$ be a classification learning algorithm with labels in $\mathcal{Y}$. We say that a (test) point $x \in \mathbb{D}^d$ is* certifiably robust *in $\mathcal{I}$ if there exists a label $\ell \in \mathcal{Y}$ such that for every possible world $I \in \mathcal{I}$, $\mathcal{L}_I(x) = \ell$. The label ℓ is called a* certain label *for x.*

In other words, suppose we call ℓ a *possible label* for x if there exists some possible world $I \in \mathcal{I}$ for which $\mathcal{L}_I(x) = \ell$, then certifiable robustness simply means that there is only one possible label for x.

Nearest Neighbor Classifiers. In k-NN classification, we are given a labeled instance I over $R(A_1, \ldots, A_d)$, along with a distance function f over $\mathbb{D}^d$. Given a point $x \in \mathbb{D}^d$, the classifier first finds the k-neighborhood $\mathcal{N}_k(x, I)$, which consists of the k points closest to x w.r.t. the distance function f. Then, the classifier assigns to x the label that is a plurality among $\mathcal{N}_k(x, I)$. When $k = 1$, the classifier returns the label of the point that is closest to x w.r.t. the distance function f. When $|\mathcal{Y}| = 2$, we are performing binary classification. We will also consider the generalization of k-NN where each tuple has a positive weight, and the classifier assigns the label with the largest total weight.

For this work, we require the following tie-breaking mechanisms: (i) if there are two labels in $\mathcal{N}_k(x, I)$ with the maximum number, then we say x is not certifiably robust for k-NN classification, and (ii) if there are more tuples with the same distance to the test point that will make $\mathcal{N}_k(x, I)$ not well-defined, we will break ties according to a predefined ordering of the tuples in the instance. By a slight abuse of notation, throughout the proof when we say $\mathcal{L}_I(x) = \ell$, we mean the number of tuples labeled ℓ is *strictly* more than that of any other labels for any choices made to pick $\mathcal{N}_k(x, I)$.

Functional Dependencies. A functional dependency (FD) is an expression of the form $X \to Y$, where X and Y are sets of attributes from R. An instance I over R satisfies $X \to Y$ if for every two tuples in I, if they agree on X they must also agree on Y. We say that I satisfies a set of functional dependencies Σ if it satisfies every functional dependency in Σ. For an attribute A and set of FDs Σ, we define $\Sigma - A$ to be the FD set where we have removed from any FD in Σ the attribute A. An *FD schema* $\mathbf{R}$ is a pair $(R(A_1, \ldots, A_d), \Sigma)$, where Σ is a set of FDs defined over R.

Repairs. Given Σ, assume that we have an inconsistent instance D that violates the functional dependencies in Σ. We say that D' is a *repair* of D if it is a maximal subset of D that satisfies Σ. In other words, we can create a repair by removing the smallest possible number of tuples from D such that all the integrity constraints are satisfied. We will use $I_\Sigma(D)$ to denote the set of all possible repairs of D w.r.t. the FD schema Σ. If the instance D is consistent, namely it does not violate any functional dependency, then $I_\Sigma(D)$ is defined to be D itself.

3.1 Problem Definitions

Although our algorithms work for any distance function such that $f(x, x')$ can be computed in time $O(1)$ (assuming the dimension d is fixed), for the hardness results we need a more precise formalization. We consider two variants of the problem. In the first variant, we will consider a specific distance function, the p-norm when the domain is $\mathbb{D} = \mathbb{R}$. Recall that for any $p \geq 1$, the p-norm is

$$\|x - x'\|_p = \left(\sum_{i=1}^{d} |x[A_i] - x'[A_i]|^p \right)^{1/p}$$

For the following definitions, we fix the dimension d and the label set $\mathcal{Y}$. Formally, we can now define the following decision problem, parameterized by an FD schema $\mathbf{R}$ and an integer $k > 0$.

▶ **Definition 3** (CR-NN$_p\langle \mathbf{R}, k \rangle$)**.** *Given an inconsistent labeled instance D over an FD schema $\mathbf{R}$ and a test point x, is x certifiably robust in $I_\Sigma(D)$ for k-NN classification w.r.t. the p-norm?*

Note that k is fixed in the above problem. We can also define the decision problem where k is instead an input parameter, denoted as CR-NN$_p\langle\mathbf{R}\rangle$.

In the second variant of the problem, instead of fixing a distance function, we will directly provide as input to the problem a strict ordering of the points in the dataset D w.r.t. their distance from the test point x. From an algorithmic point of view this does not make much difference, since we can compute the ordering in time $O(|D| \log |D|)$ for any distance function that can be computed in time $O(1)$.

▶ **Definition 4** (CR-NN$_<\langle\mathbf{R}, k\rangle$). *Given an inconsistent labeled instance D over an FD schema $\mathbf{R}$ and a strict ordering of the points in D w.r.t. their distance from a test point x, is x certifiably robust in $I_\Sigma(D)$ for k-NN classification?*

Similarly we also define the problem CR-NN$_<\langle\mathbf{R}\rangle$ with the parameter k as part of the input. Note here that there is a straightforward many-one polynomial time reduction from CR-NN$_p\langle\mathbf{R}, k\rangle$ to CR-NN$_<\langle\mathbf{R}, k\rangle$.

Finally, we define the counting variant of the problem. Given an inconsistent instance D and a label $\ell \in \mathcal{Y}$, we want to count how many repairs of D will predict label ℓ.

▶ **Definition 5** (#CR-NN$_<\langle\mathbf{R}\rangle$). *Given an inconsistent labeled instance D over an FD schema $\mathbf{R}$, a strict ordering of the points in D w.r.t. their distance from a test point x, and a label $\ell \in \mathcal{Y}$, output the number of repairs in $I_\Sigma(D)$ for which the k-NN classifier assigns label ℓ to x.*

Similarly, one can define the counting question #CR-NN$_p\langle\mathbf{R}\rangle$.

4 Main Results

In this section, we present and discuss our key results. The main dichotomy theorem relies on the notion of lhs chains for an FD schema, as defined in [21].

▶ **Definition 6** (lhs Chain). *A set of FDs Σ has a left-hand-side chain (lhs chain for short) if for every two FDs $X_1 \to Y_1$ and $X_2 \to Y_2$ in Σ, either $X_1 \subseteq X_2$ or $X_2 \subseteq X_1$.*

One can determine efficiently whether an FD schema is equivalent to one with an lhs chain or not [21].

▶ **Example 7.** Consider the relational schema $R(A, B, C, D)$. The FD set $\{A \to C, B \to C\}$ does not have an lhs-chain, since neither of the two left-hand-sides of the FDs are contained in each other. The FD set $\{AB \to C, B \to D\}$ on the other hand has an lhs chain.

We are now ready to state our main theorem.

▶ **Theorem 8** (Main Theorem). *Let $\mathbf{R}$ be an FD schema. Then, the following hold:*

- *If $\mathbf{R}$ is equivalent to an FD schema with an lhs chain, then CR-NN$_<\langle\mathbf{R}\rangle$ (and thus CR-NN$_p\langle\mathbf{R}\rangle$) is in P.*
- *Otherwise, for any integer $k \geq 1$, CR-NN$_p\langle\mathbf{R}, k\rangle$ (and thus CR-NN$_<\langle\mathbf{R}, k\rangle$) is coNP-complete.*

Moreover, it can be decided in polynomial time which of the two cases holds.

We show the polynomial time algorithm in Section 5 and the hardness proof in Section 6. We should discuss three things at this point. First, the polynomial time algorithm works for any distance function, as long as we can compute the distance between any two points in time $O(1)$. Indeed, the distance function is only needed to compute the order of tuples in the

dataset according to their distance from the test point. Second, we show the intractability result for the p-norm distance function, which is widely used in practice. It is likely that the problem remains hard for other popular distance functions as well. Third, as we will see in the next section, the tractable cases are polynomial in k, the size of the neighborhood. This is important, since k is often set to be a function of the training set size, for example $\sqrt{n}$. For the intractable cases, the problem is already hard even for $k = 1$.

Uncertain Labels. The above dichotomy theorem holds even when we allow inconsistent labels. We can model inconsistent labels by modifying the labelling function $L(t)$ to take values from $\mathcal{P}(\mathcal{Y})$, the power set of the finite label set $\mathcal{Y}$. The set of possible worlds is then defined to be the set of possible worlds of the inconsistent instance D paired with a labelling function L' such that $L'(t) \in L(t)$ for all $t \in D$. The definition of certifiable robustness carries over to this set of possible worlds.

We can simulate uncertain labels by adding an extra attribute (label) to the schema and an FD $A_1, \ldots, A_d \to \mathsf{label}$. It is easy to see that the new schema is equivalent to one with an lhs chain if and only if the original one is. Thus, we conclude that *uncertain labels do not change the complexity with respect to certifiable robustness.*

Counting. For the counting variant of certifying robustness for k-NNs, we show an analogous dichotomy result.

▶ **Theorem 9.** *Let* $\mathbf{R}$ *be an FD schema. Then, the following hold:*

- *If* $\mathbf{R}$ *is equivalent to an FD schema with an lhs chain, then* $\#CR\text{-}NN_<\langle\mathbf{R}\rangle$ *is in FP.*
- *Otherwise, even* $\#CR\text{-}NN_<\langle\mathbf{R}, 1\rangle$ *is #P-complete.*

Moreover, it can be decided in P which of the two cases holds.

5 Tractable Cases

In this section, we prove that if the FD schema $\mathbf{R}$ has an lhs chain, then there is a polynomial time algorithm for CR-NN$_<\langle\mathbf{R}\rangle$ in the size of the inconsistent dataset, the parameter k and the number of possible labels. Then, we show that when $\mathbf{R}$ consists of a single primary key we can construct an even faster algorithm that runs in *linear time* w.r.t the number of tuples, number of labels, and k.

For this section, let D be an inconsistent labeled instance and x be the test point. Assume w.l.o.g. that $\mathcal{Y} = \{1, 2, \ldots, m\}$ and let n be the number of tuples in D. We assume that the points in D are already in strict order w.r.t. their distance from x: $t_1 < t_2 < \cdots < t_n$.

5.1 An Algorithm for Certifiable Robustness

Note that in the following analysis we fix an FD schema $\mathbf{R}$ with an lhs chain. The algorithm first constructs a repair I by choosing greedily points using the given ordering as long as they do not conflict with each other. This step can be implemented in time $O(n)$ by, say, building a hash map per FD which maps for each tuple the value of the LHS attribute(s) to the value of the RHS attribute(s). Suppose w.l.o.g. that $\mathcal{L}_I(x) = 1$.

As a second step, for every label $\ell \in \{2, \ldots, m\}$, we will attempt to construct a repair I' of D such that the number of ℓ-labeled points is at least as many as the number of 1-labeled points in the k-neighborhood of x. Such a repair will be a witness that some other label is possible for x, hence x is not certifiably robust.

It will be helpful now to define the following terms for a subinstance $I \subseteq D$, $\tau \in \{1, \ldots, n\}$, and a label $\ell \in \mathcal{Y}$:

$$\mathcal{N}_\tau^{\leq}(I) = \{t_j \in I \mid j \leq \tau\}$$
$$C_\tau^{\leq}(\ell, I) = |\{t \in \mathcal{N}_\tau^{\leq}(I) \mid L(t) = \ell\}|$$

We are now ready to present the core component of our algorithm. This component will be executed for every label $\ell > 1$ and $\tau \in \{1, \ldots, n\}$. Thus, it will run $O(|\mathcal{Y}| \cdot n)$ times. Define the following quantity for a subinstance $J \subseteq D$, an FD set Δ, and i where $0 \leq i \leq k$:

$$M_i[J, \Delta] = \max\{C_\tau^{\leq}(\ell, K) - C_\tau^{\leq}(1, K) \mid K \in I_\Delta(J) \text{ s.t. } |\mathcal{N}_\tau^{\leq}(K)| = i\}.$$

Here for simplicity we adopt a slight abuse of notation where, although $M_i[J, \Delta]$ depends on τ, τ is not explicitly shown in the notation $M_i[J, \Delta]$. If there is no repair K for J such that $|\mathcal{N}_\tau^{\leq}(K)| = i$, we define $M_i[J, \Delta] = -\infty$. The key observation is that if $M_k[D, \Sigma] \geq 0$ then ℓ is a possible label for x and hence robustness is falsified. The algorithm computes this quantity using a combination of dynamic programming and recursion on the structure of the FD set.

The Recursive Algorithm. Given $J \subseteq D$ and a set of FDs Δ, we want to compute $M_i[J, \Delta]$ for every $i = 0, \ldots, k$. First, we remove from Δ any trivial FDs. Then we distinguish three cases:

Base Case: *the set of FDs is empty.* In this case, $I_\Delta(J) = \{J\}$. For every $i \neq |\mathcal{N}_\tau^{\leq}(J)|$, $M_i[J, \Delta] = -\infty$. For $i = |\mathcal{N}_\tau^{\leq}(J)|$ (if $|\mathcal{N}_\tau^{\leq}(J)| \leq k$), we have $M_i[J, \Delta] = C_\tau^{\leq}(\ell, J) - C_\tau^{\leq}(1, J)$, so we can compute this in a straightforward way.

Consensus FD: *there exists an FD $\emptyset \to A$.* In this case, we recursively call the algorithm to compute $M_i[\sigma_{A=a}(J), \Delta - A]$ for every $a \in \pi_A(J)$. Then, for every $i = 0, \ldots, k$:

$$M_i[J, \Delta] = \max_{a \in \pi_A(J)} M_i[\sigma_{A=a}(J), \Delta - A]$$

Common Attribute: *there exists a common attribute A in the lhs of all FDs.* In this case, we recursively call the algorithm to compute $M_i[\sigma_{A=a}(J), \Delta - A]$ for every $a \in \pi_A(J)$. Let $S = \pi_A(J) = \{a_1, \ldots, a_{|S|}\}$. Then, for every $i = 0, \ldots, k$:

$$M_i[J, \Delta] = \max_{\sum_{a \in S} i_a = i} \sum_{a \in S} M_{i_a}[\sigma_{A=a}(J), \Delta - A]$$

We next discuss how to do the above computation using dynamic programming. We can view $M_{i_a}[\sigma_{A=a}(J), \Delta - A]$ as a matrix with rows indexed by value a of attributes A and columns indexed by i_a with $0 \leq i_a \leq k$. The task is to pick one entry from each row so that the sum of entries is maximized and the column indices of entries sum to i. The dynamic programming computes an $|S| \times (k+1)$ matrix $\mathcal{M}$ where the (i, j)-entry represents the maximum of $\sum_{u=1}^{i} M_{i_u}[\sigma_{A=a_u}(J), \Delta - A]$ such that $\sum_{u=1}^{i} i_u = j$ and finally returns the entries $\mathcal{M}[|S|, i]$.

The algorithm runs in polynomial time with respect to the size of D, the parameter k and the number of labels $|\mathcal{Y}|$. For detailed analysis on the correctness of the algorithm and its running time, see [7].

Weighted Majority. The algorithm can also handle the case where we compute the predicted label by weighted majority, where each tuple t is assigned a weight w_t. The only thing we need to modify is the definition of $C_\tau^{\leq}(\ell, I)$, which now becomes $\sum_{t \in \mathcal{N}_\tau^{\leq}(I): L(t) = \ell} w_t$.

Algorithm 1 Dynamic Programming.

```
for j = 0, ..., k do
    M[1, j] ← M_j[σ_{A=a_1}(J), Δ − A] ;
for i = 2, ..., |S| do
    for j = 0, ..., k do
        M[i, j] ← max_u{M[i − 1, u] + M_{j−u}[σ_{A=a_i}(J), Δ − A]} ;
```

(a) Ordering of the tuples in instance D.

(b) Result of $\textsc{Prune}(D, 2, 1)$.

(c) Result of $\textsc{Prune}(D, 3, 1)$.

Figure 1 Running example for the single primary key algorithm. Tuples with the same color/shape belong in the same block.

5.2 A Faster Algorithm for Single Primary Key

The algorithm given in the above section, though in polynomial time, is not very efficient. In this section, we give a faster algorithm for certifiable robustness when the FD schema is equivalent to one with a single primary key. Recall that in this case we need to pick exactly one tuple from the set of tuples that share the same key.

As in the previous section, we will first run k-NN on an arbitrarily chosen repair to obtain a possible label for x (this part needs only linear time). Without any loss of generality, assume that the predicted label for x is 1. For every target label $\ell \in \{2, \ldots, m\}$, we will attempt to construct a repair such that ℓ occurs at least as many times as 1 in the k-neighborhood of x. If such a repair exists, then robustness is falsified.

For a tuple t, we denote $\mathsf{key}(t)$ to be its key. The *block* of a tuple t is the set of tuples with the same key. Further, define $C(\ell, I) = |\{t \in \mathcal{N}_k(x, I) \mid L(t) = \ell\}|$.

Suppose now we want to find a repair I such that $C(\ell_2, I) \geq C(\ell_1, I)$. Define $\textsc{Prune}(D, \ell_2, \ell_1)$ to be the dataset obtained from D if we remove any tuple $t \in D$ such that there exists another tuple $t' \in D$ in the same block with:

1. $t < t'$ and $L(t) = \ell_1$; or
2. $t > t'$ and $L(t') = \ell_2$.

▶ **Lemma 10.** *Let $\ell_1, \ell_2 \in \mathcal{Y}$ and $D^\star = \textsc{Prune}(D, \ell_2, \ell_1)$. Then, there exists a repair I of D s.t. $C(\ell_2, I) \geq C(\ell_1, I)$ if and only if there exists a repair I' of $D^\star$ with $C(\ell_2, I') \geq C(\ell_1, I')$.*

Lemma 10 tells us that it suffices to consider $D^\star$ instead of D. For the proof, see [7]. $D^\star$ has the following nice properties:

- every block has at most one tuple with a label from $\{\ell_1, \ell_2\}$.
- any tuple with label in $\{\ell_1, \ell_2\}$ is always the last tuple in its block (i.e. the one furthest away from x).

The pruning procedure can be implemented in linear time $O(n)$. Algorithm 2 FastScan now attempts to find the desired repair. It runs in linear time with respect to the size of D and the number of labels $|\mathcal{Y}|$ and, moreover, its time complexity does not depend on k. For detailed analysis on the correctness of the algorithm and its running time, see [7].

Algorithm 2 FastScan.

Input: instance D, test point x, labels ℓ_1, ℓ_2
Output: whether there exists repair I s.t. $C(\ell_2, I) \geq C(\ell_1, I)$

$D \leftarrow \text{Prune}(D, \ell_2, \ell_1)$;
$n_1, n_2 \leftarrow 0$;
$\mathsf{B}, \mathsf{B}^{\square} \leftarrow \{\}$;
for $i \leftarrow 1$ **to** $|D|$ **do**
  $\mathsf{B} \leftarrow \mathsf{B} \cup \{\mathsf{key}(t_i)\}$;
  if $L(t_i) = \ell_2$ **then**
    $n_2 \leftarrow n_2 + 1$;
  if *t_i is the only tuple of its block and* $L(t_i) = \ell_1$ **then**
    $n_1 \leftarrow n_1 + 1$;
  if *t_i is the last tuple of its block* **then**
    $\mathsf{B}^{\square} \leftarrow \mathsf{B}^{\square} \cup \{\mathsf{key}(t_i)\}$;
  if $|B^{\square}| \leq k \leq |B|$ *and* $n_2 \geq n_1$ **then**
    return true;
return false;

▶ **Theorem 11.** *There exists an $O(|\mathcal{Y}|n)$ algorithm for CR-NN$_<\langle\mathbf{R}\rangle$ when the FD schema $\mathbf{R}$ is equivalent to one with a single primary key.*

When $|\mathcal{Y}| = 2$, the algorithm essentially reduces to the MinMax algorithm in [13]. For $|\mathcal{Y}| \geq 3$ it outperforms the SortScan algorithm [13], since the latter algorithm has an exponential dependence on $|\mathcal{Y}|$ and k. Our algorithm also can deal with the case where two tuples in the same block have different labels, which is not something the MinMax and SortScan algorithms can handle.

▶ **Example 12.** We now illustrate our algorithm by a simple example where $k = 3$ and $\mathcal{Y} = \{1, 2, 3\}$. Figure 1a represents an inconsistent instance D, where nodes with the same shape/color are in the same block. Their distances to a given test point are increasing from left to right. A repair that chooses the first two tuples assigns label 1 to x, hence 1 is a possible label. Figures 1b and 1c illustrate the pruned instances Prune(D, 2, 1) and Prune(D, 3, 1), respectively. Take Figure 1c for example: when FastScan reaches the iteration where $i = 3$, we have $n_2 = 2, n_1 = 1$, $|B^{\square}| = 3$ and $|B| = 3$, so $|B^{\square}| = k = |B|$ and $n_2 \geq n_1$. Indeed, by choosing the first, second, third, and last two tuples in Figure 1c, we see that label 1 is not robust (against label 3).

6 Hardness Result

In this section, we establish the main intractability result.

▶ **Theorem 13.** *Let $\mathbf{R}$ be an FD schema that is not equivalent to any FD schema with an lhs chain. Then, the problem CR-NN$_p\langle\mathbf{R}, 1\rangle$ is coNP-complete for any $p > 1$.*

Proof. The coNP membership of the problem CR-NN$_p\langle \mathbf{R}, 1\rangle$ follows from the observation that if one is not certainly robust, then it can be checked efficiently two given repairs (certificate) indeed lead to two different prediction labels. To prove this hardness result, we will describe a reduction from the SAT-3-restricted problem, inspired by the construction of [27] for the edge dominating set problem. In this variant of SAT, each clause has at most three literals, while every variable occurs two times and its negation once.

Let ϕ be a SAT-3-restricted formula. Suppose that ϕ has m clauses $C_1, C_2, \ldots, C_m$ and n variables $x_1, x_2, \ldots, x_n$. Our construction consists of three steps.

Step 1. In the first step, we construct a directed labeled graph $G = (V, E)$ with labels in $\{0, 1\}$:

- The set of vertices $V = \{C_i \bigcup x_j^k \bigcup y_j^k$ where $1 \le i \le m, 0 \le k \le 2$ and $1 \le j \le n\}$.
- For each clause C_i, where $i = 1, \ldots, m$, we add the following labeled edge:

  $$(C_i^+, C_i^-) \to 0$$

  That is, we add the directed edge which points from C_i^+ to C_i^- to the set of edges E and label it as 0.
- Suppose that variable x_j, where $j = 1, \ldots, n$, appears positive in clauses C_κ, C_λ, and negative in clause C_μ. Then, we introduce the following labeled edges:

  $$(C_\kappa^+, x_j^0), (y_j^0, x_j^0) \to 1$$
  $$(C_\lambda^+, x_j^1), (y_j^1, x_j^1) \to 1$$
  $$(x_j^2, C_\mu^-), (x_j^2, y_j^2) \to 1$$
  $$(y_j^0, y_j^2), (y_j^1, y_j^2) \to 0$$

Figure 2 shows the above construction. Note that G is a directed bipartite graph, since no vertex has both incoming and outgoing edges. Hence, one can equivalently view each maximal matching of G as a subset repair of an instance with FD schema $(R(A, B), \{A \to B, B \to A\})$ and vice versa (attributes A and B correspond to the two sides of the bipartite graph).

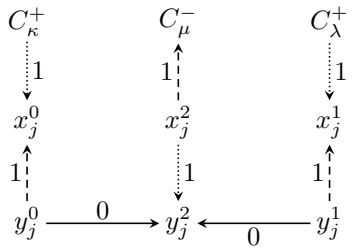


Figure 2 Variable gadget for the hardness reduction from the SAT-3-restricted problem.

▷ Sublemma 1. ϕ is satisfiable if and only if there exists a maximal matching for G that includes only edges with label 1.

Proof of Sublemma 1. ⇒ Assume that the variable assignment ψ makes ϕ satisfiable. Fix any order of variables $x_1 \ldots, x_n$. We form a set of edges M as follows. For any variable x_j visited in the above order, we distinguish two cases:

- $\psi(x_j) =$ true: we pick (x_j^2, y_j^2). If C_κ^+ is not yet matched, pick (C_κ^+, x_j^0), otherwise pick (y_j^0, x_j^0). Similarly for C_λ^+.
- $\psi(x_j) =$ false: pick (y_j^0, x_j^0) and (y_j^1, x_j^1). If C_μ^- is not yet matched, pick (x_j^2, C_μ^-), otherwise pick (x_j^2, y_j^2).

By construction, M contains only edges with label 1.

▷ Claim 1 (M is a matching). Since x_j, y_j occur only in a variable gadget, they will have at most one adjacent edge from M. By construction, each clause C_κ^+, C_μ^- will also have at most one adjacent edge from M.

▷ Claim 2 (M is a maximal matching). First, observe that by our construction, all x_j^0, x_j^1, x_j^2 are matched for any $j = 1, \ldots, n$. Second, notice that if $\psi(x_j) = \mathsf{true}$, the edge (x_j^2, y_j^2) will be chosen; if $\psi(x_j) = \mathsf{false}$, the edges (y_j^0, x_j^0) and (y_j^1, x_j^1) will be chosen. Thus, the edges $(y_j^0, y_j^2), (y_j^1, y_j^2)$ can not be added to M. Finally, consider the edge (C_i^+, C_i^-) corresponding to clause C_i. If there exists a positive literal which satisfies C_i, then consider the earliest x_j in the linear order of variables. By construction, (C_i^+, x_j^ν) is in the matching, where $\nu \in \{0, 1\}$. Otherwise, C_i is satisfied by a negative literal: consider the earliest such x_k in the linear order. Then (x_k^2, C_i^-) is in M. In either case, (C_i^+, C_i^-) cannot be added to increase the size of the matching.

⇐ Assume a maximal matching M that avoids 0-labeled edges. For every variable x_j, if M contains (x_j^2, y_j^2), we assign $\psi(x_j) = \mathsf{true}$, otherwise $\psi(x_j) = \mathsf{false}$. We claim that ψ is a satisfying assignment for ϕ. Indeed, take any clause C_i. Since $(C_i^+, C_i^-) \notin M$, there exists some edge in M that conflicts with it. If this edge is of the form (C_i^+, x_j^ν), then it must be that $(x_j^2, y_j^2) \in M$. But then $\psi(x_j) = \mathsf{true}$, and since x_j occurs positively in C_i the clause is satisfied. If this edge is of the form (x_j^2, C_i^-), then $(x_j^2, y_j^2) \notin M$. Thus, $\psi(x_j) = \mathsf{false}$, and since x_j occurs negatively in C_i it is again satisfied. ◁

Step 2. A maximal matching of G can be viewed equivalently as a repair of a labeled instance D_0 with FD schema $\mathbf{S} = (R(A, B), \{A \to B, B \to A\})$. In the second step, we will transform the instance D_0 of $\mathbf{S}$ to a labeled instance D_1 of the target FD schema $\mathbf{R}$. We will do this using the concept of *fact-wise reductions*. A fact-wise reduction from $\mathbf{S}$ to $\mathbf{R}$ is a function Π that maps a tuple from an instance of $\mathbf{S}$ to a tuple of an instance of $\mathbf{R}$ such that (i) it is injective, (ii) it preserves consistency and inconsistency (i.e. a tuple in D_0 violates $\mathbf{S}$ if and only if the corresponding tuple in D_1 violates $\mathbf{R}$), and (iii) it can be computed in polynomial time. In fact, we will use exactly the same fact-wise reduction as the one used in [21] to reduce an instance in $\mathbf{S}$ to one in $\mathbf{R}$, where $\mathbf{R}$ is not equivalent to an FD schema with an lhs chain. It will be necessary to present this reduction in detail, since its structure will be needed for the third step of our reduction.

W.l.o.g., we can assume the FD schema is minimal. Since it does not have an lhs chain, there are two FDs $X \to A$ and $X' \to A'$ such that $X \subsetneq X'$ and $X' \subsetneq X$. Let $\oplus$ be a fresh constant. We map $t = R(u, v)$ with label ℓ to a tuple $\Pi(t)$ with label also ℓ such that:

$$\Pi(t)[A_i] = \begin{cases} \oplus & \text{if } A_i \in (X \cap X')^{+,\Sigma} \\ u & \text{if } A_i \in X \setminus (X \cap X')^{+,\Sigma} \\ v & \text{if } A_i \in X' \setminus (X \cap X')^{+,\Sigma} \\ (u, v) & \text{otherwise.} \end{cases}$$

Here, $X^{+,\Sigma}$ denotes the closure of an attribute set X w.r.t. the FD set Σ. By [21] we know that Π is a fact-wise reduction. Let D_1 be the resulting instance of $\mathbf{R}$.

▷ Sublemma 2. ϕ is satisfiable if and only if there exists a repair for D_1 in $\mathbf{R}$ that includes only tuples with label 1.

Proof of Sublemma 2. This follows from Sublemma 1 and the fact that Π is a fact-wise reduction. ◁

Step 3. In the last step of the reduction, we will present an encoding $[\![\cdot]\!]$ that embeds the values of the attributes in D_1 to values in $\mathbb{N}$, hence allowing us to compute distances with the p-norm. The resulting tuples will also be labeled from $\mathcal{Y} = \{0, 1\}$.

Let $\alpha = d \cdot (2m + 8n)$, where d is the number of attributes. First, let $[\![\oplus]\!] = 0$. For a vertex $u \in V$, let

$$[\![u]\!] = \begin{cases} i & \text{if } u = C_i^+ \\ m + i & \text{if } u = C_i^- \\ 2m + 3j + \nu & \text{if } u = y_j^\nu \\ \alpha + 3j + \nu & \text{if } u = x_j^\nu \end{cases}$$

It is easy to see that the above embedding is injective, meaning that if $[\![u]\!] = [\![v]\!]$ then $u = v$. As for the edges, consider any ordering $e_1, e_2, \ldots$ and simply let $[\![e_i]\!] = i$. Note that the number of edges in G is $|E| = 8n + 2m$. This embedding is also injective. Let $D_2 = [\![D_1]\!]$ denote the instance we obtain by encoding every value of D_1 as above. Since the encoding is injective, this is also trivially a fact-wise reduction, hence Sublemma 2 holds for D_2 as well.

▷ Sublemma 3. ϕ is satisfiable if and only if $x = R(0, 0, \ldots, 0)$ has no certain label in D_2.

Proof of Sublemma 3. We first need the following two claims.

▷ Claim 1. *Any tuple with label 0 is closer to x than any tuple with label 1.* Indeed, a tuple has label 1 if and only if it contains in an attribute a value of the form $[\![x_j^\nu]\!]$. Hence, any tuple with label 1 has distance $> \alpha$ from x. On the other hand, each attribute in a 0-labeled tuple has value at most $2m + 8n$. Hence, the distance from any tuple with label 0 is bounded by $d^{1/p} \cdot (2m + 8n) \leq d \cdot (2m + 8n)$.

▷ Claim 2. *0 is a possible label for x.* Indeed, the tuple corresponding to the edge (C_1^+, C_1^-) is the closest one to x and has label 0. Hence, any repair that includes this tuple will assign the label 0 to x.

⇒ Assume that the variable assignment ψ makes ϕ satisfiable. Then, we know that there exists a repair for D_2 that avoids any tuple with label 0. This repair will then assign label 1 to x, which implies that x is not a certain label since by **Claim 2** 0 is a possible label for x.

⇐ Assume a repair that assigns a label 1 to x – hence making x have no certain label. Since by **Claim 1** all 0-labeled tuples are closer than the 1-labeled tuples, this means that all tuples in the repair must have label 1. But then, ϕ is satisfiable. ◁

This completes the proof. ◀

Finally, we extend the intractability result from CR-NN$_p\langle \mathbf{R}, 1\rangle$ to CR-NN$_p\langle \mathbf{R}, k\rangle$ for any integer $k \geq 1$. For the proof, see [7].

▶ **Theorem 3.** *Let $\mathbf{R}$ be an FD schema that is not equivalent to any FD schema with an lhs chain. Then, for any integer $k \geq 1$, CR-NN$_p\langle \mathbf{R}, k\rangle$ is coNP-hard for any $p > 1$.*

7 Optimal Repairs Revisited

In this section, we investigate the complexity landscape of a related problem to certifying robustness for k-NN classification, which may be of independent interest. In [20], the authors studied the Opt-Repair problem: each tuple t is associated with a positive weight $w_t \geq 0$,

and we want to find the optimal subset repair that removes the set of tuples with the smallest total weight. Note that this is equivalent to finding the repair with the largest total weight.

Here, we consider the symmetric problem, MIN-REPAIR: we want to find *the subset repair that has tuples with the smallest total weight.* In this case, we interpret the tuple weights as a measure of how "wrong" we think a tuple is. We can parametrize this problem with a given FD schema $\mathbf{R}$, as in MIN-REPAIR$\langle\mathbf{R}\rangle$. MIN-REPAIR captures as a special case the following problem, denoted as FORBIDDEN-REPAIR$\langle\mathbf{R}\rangle$: given an inconsistent instance D over $\mathbf{R}$ and a subinstance $S \subseteq D$, does there exist a subset repair $I \subseteq D$ such that $I \cap S = \emptyset$?

▶ **Lemma 4.** *There exists a many-one polynomial time reduction from* FORBIDDEN-REPAIR$\langle\mathbf{R}\rangle$ *to* MIN-REPAIR$\langle\mathbf{R}\rangle$.

Proof. One can set the weight of any tuple in S to be 1, otherwise 0. Then, there exists a repair that avoids the forbidden set S if and only if there exists a repair with total weight equal to 0. ◀

From the hardness proof of Theorem 13, we immediately obtain the following intractability result.

▶ **Theorem 5.** *Let* $\mathbf{R}$ *be an FD schema that is not equivalent to any FD schema with an lhs chain. Then,* FORBIDDEN-REPAIR$\langle\mathbf{R}\rangle$ *is NP-hard. As a result,* MIN-REPAIR$\langle\mathbf{R}\rangle$ *is also NP-hard.*

It turns out that the forbidden set repair problem is directly connected with certifying robustness for 1-NN classification.

▶ **Lemma 6.** *There exists a many-one polynomial time reduction from* FORBIDDEN-REPAIR$\langle\mathbf{R}\rangle$ *to the complement of* CR-NN$_<\langle\mathbf{R}, 1\rangle$.

Proof. Let D, $S \subseteq D$ be the inputs to the FORBIDDEN-REPAIR problem. We will construct a labeled instance for the classification problem using only two labels, $\mathcal{Y} = \{0, 1\}$. The input instance is exactly D. For labeling, if $t \in S$ then $L(t) = 0$, otherwise $L(t) = 1$. Finally, we construct an ordering of the tuples in D such that $t < t'$ whenever $t \in S$, $t' \in D \setminus S$.

We first claim that any repair of D that avoids S is a repair that assigns a label of 1 to x. Indeed, the 1-neighborhood of x for such a repair will consist of a tuple with label 1 by construction. On the other hand, any repair that includes a tuple from S assigns label 0 to x. Since there exists at least one repair includes a tuple from S, there exists no repair that avoids the forbidden set S if and only if x is certifiably robust (with label 0). ◀

The above reduction gives a polynomial time algorithm for FORBIDDEN-REPAIR$\langle\mathbf{R}\rangle$ whenever $\mathbf{R}$ is equivalent to an FD schema with an lhs chain. We now present Algorithm 3 that works for the more general MIN-REPAIR problem.

The algorithm works similarly to the OptSRepair algorithm in [20] with two differences. First, we only need to consider the cases where the FD schema has a common lhs or a consensus FD (i.e., FD with an empty lhs). Second, in the case of the consensus FD, where we partition the instance, we take the repair that has the *minimum* total weight instead of the largest one.

▶ **Theorem 7.** *Let* $\mathbf{R}$ *be an FD schema that is equivalent to an FD schema with an lhs chain. Then,* MIN-REPAIR$\langle\mathbf{R}\rangle$ *is in P.*

Algorithm 3 MIN-REP(Σ, D).

if Σ *is trivial* **then**
  return D
remove trivial FDs from Σ ;
if Σ *has a common lhs attribute* A **then**
  return $\cup_{a \in \pi_A(D)}$MIN-REP$(\Sigma - A, \sigma_{A=a}(D))$
if Σ *has a consensus FD* $\emptyset \to A$ **then**
  $m \leftarrow \arg\min_{a \in \pi_A(D)} w(\text{MIN-REP}(\Sigma - A, \sigma_{A=a}(D)))$;
  return MIN-REP$(\Sigma - A, \sigma_{A=m}(D))$

The dichotomy we obtain for MIN-REPAIR coincides with the one we obtain for CR-NN. However, it is different from the dichotomy observed for the OPT-REPAIR problem in [20]; in fact, every FD schema that admits a polynomial time algorithm for MIN-REPAIR also admits a polynomial time algorithm for OPT-REPAIR, but not vice versa. Specifically, the FD schema $(R(A, B), \{A \to B, B \to A\})$ is hard for MIN-REPAIR, but tractable for OPT-REPAIR. The reason is that finding a maximum-weight (maximal) matching in a bipartite graph is polynomially solvable, but finding a minimum-weight maximal matching is NP-hard.

8 Certifiable Robustness by Counting

In this section, we consider the counting version of certifiable robustness for k-NN classifiers. The counting problem of certifiable robustness asks the following: among all possible repairs, how many will predict label ℓ for a fixed $\ell \in \mathcal{Y}$?

We show that the counting problem still remains in polynomial time if the FD set $\mathbf{R}$ is equivalent to an lhs chain by generalizing the algorithm in Section 5.1. Formally, the class of counting problems that can be computed in polynomial time is called FP.

▶ **Theorem 8.** *If the FD schema* $\mathbf{R}$ *is equivalent to some FD schema with an lhs chain, then the counting problem* $\#CR\text{-}NN_<\langle \mathbf{R} \rangle$ *is in FP.*

We show now how to generalize the algorithm in Section 5.1 to perform counting. Let x be the test point and $\mathcal{Y} = \{1, , 2 \ldots, m\}$. It suffices to show how to count for the label $\ell = 1$. Recall the definition of $\mathcal{N}_\tau^{\leq}(x, I)$ and $C_\tau^{\leq}(\ell, I)$. Similarly as in Section 5.1, we will compute a high-dimensional matrix M "layer by layer" and read off the answer from it.

We now make our key definition. Fix a threshold $\tau > 0$, and define the following quantity for a (possibly inconsistent) subinstance $J \subseteq D$ and integers $i, c_2, c_3, \ldots, c_m$, where $0 \leq i \leq k$ and $-k \leq c_j \leq k$ for all $j \in \{2, 3, \ldots, m\}$:

$$M_i[J, c_2, \ldots, c_m, \Delta] = \{\#K \mid K \in I_\Delta(J) \text{ s.t. } |\mathcal{N}_\tau^{\leq}(x, K)| = i \text{ and } C_\tau^{\leq}(j, K) - C_\tau^{\leq}(1, K) = c_j \ \forall j \in \{2, 3, \ldots, m\}\}.$$

For simplicity of notation, let $\mathbf{c}$ denote the vector $(c_2, c_3, \ldots, c_m)$ with the understanding that $\mathbf{c}_j$ could represent any new vector $(c_{2_j}, c_{3_j}, \ldots, c_{m_j})$. Sometimes we might suppress the FD set Δ to write $M_i[J, \mathbf{c}]$ when the context is clear. The interpretation for the entry $M_i[J, \mathbf{c}]$ is that it records the number of repairs in J with i many tuples which are among τ-th closest to x such that the differences between the number of 1-tuples and the number of j-tuples are exactly c_j, $j \in \{2, 3, \ldots, m\}$. If there is no such $K \in I_\Sigma(J)$, we define $M_i[J, \mathbf{c}] = 0$. The algorithm computes the entries of M by a combination of dynamic programming and

recursion on the FD schema $\mathbf{R}$. Note that after computing M, the answer to $\#\text{CR-NN}_<\langle\mathbf{R}\rangle$ is exactly the sum of all entries $M_k[D, \mathbf{c}]$ where $c_j \geq 1$ for all $j \in \{2, 3, \ldots, m\}$. The algorithm has three disjoint cases when computing $M_i[J, \mathbf{c}]$:

Base Case: *the set of FDs is empty.* In this case, $M_i[J, \mathbf{c}] = 0$ unless $|\mathcal{N}_\tau^{\leq}(x, I)| = i \leq k$ and $C_\tau^{\leq}(j, J) - C_\tau^{\leq}(\ell, J) = c_j$ for all $j \in \{2, 3, \ldots, m\}$, in which case the entry is 1. This step clearly can be computed efficiently.

Consensus FD: *there exists an FD $\emptyset \to A$.* In this case, we recursively call the algorithm to compute $M_i[\sigma_{A=a}(J), \mathbf{c}, \Delta - A]$ for every value $a \in \pi_A(J)$. Then, we calculate $M_i[J, \mathbf{c}] = \sum_{a \in \pi_A(J)} M_i[\sigma_{A=a}(J), \mathbf{c}, \Delta - A]$.

Common Attribute: *there exists a common attribute A in the lhs of all FDs.* In this case, we recursively call the algorithm to compute $M_j[\sigma_{A=a}(J), \mathbf{c}_j, \Delta - A]$ for every value $a \in \pi_A(J)$ and all $j, \mathbf{c}_j$ such that $0 \leq j \leq i$ and $0 \leq c_{l_j} \leq c_l$ where $2 \leq l \leq m$. Let $S = \pi_A(J) = \{a_1, \ldots, a_{|S|}\}$. We then compute

$$M_i[J, \mathbf{c}] = \sum \sum_{j=1}^{|S|} M_{i_j}[\sigma_{A=a_j}(J), \mathbf{c}_j, \Delta - A]$$

where the first summation is over all possible solutions of i_j's and c_{l_j}'s such that $\sum_{j=1}^{|S|} i_j = i$ and $\sum_{j=1}^{|S|} c_{l_j} = c_l$ for all $l \in \{2, 3, \ldots, m\}$. We now show how to compute $M_i[J, \mathbf{c}]$ by a direct generalization of Algorithm 1. The dynamic programming computes a $(m+1)$-dimensional matrix $\mathcal{M}$ where its $(p, q, \mathbf{c}_i)$-entry represents the sum $\sum_{j=1}^{p} M_{i_j}[\sigma_{A=a_j}(J), \mathbf{c}_j]$ over all possible solutions i_j's and $\mathbf{c}_j$'s such that $\sum_{j=1}^{p} i_j = q$ and $\sum_{j=1}^{p} c_{l_j} = c_{l_i}$ for $2 \leq l \leq m$. Note that $\mathcal{M}$ has $|S| \cdot (k+1) \cdot (2k+1)^{m-1} = O(n \cdot k^m)$ entries. Let $K = \{k, k-1, \ldots, -k\}$ and $\mathbf{c}_i - \mathbf{c}_j := (c_{2_i} - c_{2_j}, \ldots, c_{m_i} - c_{m_j})$. We are now ready to present our dynamic programming algorithm:

Algorithm 4 Dynamic Programming (Counting).

```
for j = 0, ..., k and c ∈ K^m do
    M_j[1, c] ← M_j[σ_{A=a_1}(J), c, Δ − A] ;
for i = 2, ..., |S| do
    for j = 0, ..., k and c ∈ K^m do
        M_j[i, c] ← Σ_{p, c_q} {M_p[i − 1, c_q] + M_{j−p}[σ_{A=a_i}(J), c − c_q, Δ − A]};
```

We show in [7] that the counting algorithm runs in polynomial time with respect to the size of D and the parameter k. However, the running time depends exponentially on the number of labels $|\mathcal{Y}|$. An open question remains whether this exponential dependency is essential.

The hardness part for counting is an immediate corollary with the previous result in [21].

▶ **Theorem 9.** *If the FD schema $\mathbf{R}$ is not equivalent to some FD schema with an lhs chain, then $\#$CR-NN$_<\langle\mathbf{R}\rangle$ is $\#$P-complete.*

Proof. By Theorem 3.2 in [21], it is $\#$P-hard to count the number of repairs for the FD schema $\mathbf{R}$. Now, given any instance D, we can pick any ordering of the points and assign the same label ℓ to every tuple. Then, the number of repairs that predict label ℓ is the same as the total number of repairs. ◀

The approximate counting of certifying robustness of k-NN classifiers is also hard.

▶ **Theorem 10.** *If the FD schema* $\mathbf{R}$ *is not equivalent to some FD schema with an lhs chain, then* $\#CR\text{-}NN_<\langle\mathbf{R}\rangle \equiv_{AP} \#SAT$.

Here, $\equiv_{AP}$ refers to the equivalency up to approximation-preserving reductions [6].

Proof. By the same fact-wise reduction in Lemma 2 and labelling in Theorem 9, we have $\#\textsc{MaximalBIS} \leq \#\textsc{SRep} \leq \#\text{CR-NN}_<\langle\mathbf{R}\rangle$ where $\#\textsc{MaximalBIS}$ is the problem of counting the number of maximal independent sets in a bipartite graph and $\#\textsc{SRep}$ is the problem of counting the number of subset repairs of an inconsistent instance. By Theorem 1 in [9], $\#\textsc{MaximalBIS} \equiv_{\text{AP}} \#\text{SAT}$ and thus we obtain $\#\text{CR-NN}_<\langle\mathbf{R}\rangle \equiv_{AP} \#\text{SAT}$. ◀

9 Other Uncertainty Models

In this section, we study the complexity of certifying robustness for k-NNs under three simple uncertainty models: ?-sets, or-sets, and Codd tables. We show that for these models we can certify robustness in polynomial time. Throughout the section, we fix the relational schema to be $R(A_1, \dots, A_d)$.

?-Sets with Size Constraints. For a given instance D over the schema, we mark an uncertain subset $D_?$ of the tuples in D. Then, for a positive integer $m \geq 1$, we define the set of possible worlds as:

$$\mathcal{I}_? = \{I \mid D \setminus D_? \subseteq I \subseteq D, |D \setminus I| \leq m\}.$$

In other words, we can construct a possible world by removing any – but at most m – tuples from $D_?$. When $m = |D_?|$, this definition captures the notion of ?-tables as defined in [25]. When $D_? = D$, it captures *data poisoning* scenarios, where the goal is to protect against attacks that can poison up to m tuples of the training set.

For this setting, we can show that certifiable robustness for k-NNs can be computed in almost linear time in the size of the input.

▶ **Lemma 11.** *We can certify robustness in* $\mathcal{I}_?$ *for* k*-NNs in time* $O((|\mathcal{Y}| + \log n) \cdot n)$ *where* n *is the size of the dataset.*

Proof. For the sake of simplicity, let $\mathcal{Y} = \{1, 2, \dots\}$.

Let x be the test point. As a first step, we order the tuples in increasing order of $f(x, t)$ – this can be done in time $O(n \log n)$. We assume w.l.o.g. that all distances are distinct. Let the resulting order be $t_1, t_2, \dots, t_n$.

Since it always holds that $D \in \mathcal{I}_?$, we can fist compute the predicted label of x in D in time $O(k)$. W.l.o.g. assume that $\mathcal{L}_D(x) = 1$. For every label $\ell > 1$, the algorithm will now try to construct a possible world where ℓ occurs at least as many times as 1 in the k-neighborhood of x. To make the algorithm exposition easier, define for every tuple $t \in D_?$ a priority value $\rho(t)$ as follows.

$$\rho(t) = \begin{cases} 2, & \text{if } L(t) = \ell \\ 0, & \text{if } L(t) = 1 \\ 1, & \text{otherwise.} \end{cases}$$

Algorithm 5 Certifiable Robustness for ?-Sets.

Input: test point x
Output: is k-NN certifiably robust in $\mathcal{I}_?$

$J \leftarrow \{t_1, \ldots, t_k\}$;
$\Delta \leftarrow 0$;
for $i = k+1$ **to** n **do**
  $\Delta \leftarrow \Delta + 1$;
  if $|\{t \in J \mid L(t) = \ell\}| \geq |\{t \in J \mid L(t) = 1\}|$ **then**
    return true;
  if $J \cap D_? = \emptyset$ *or* $\Delta > m$ **then**
    return false;
  $J \leftarrow J \cup \{t_i\}$;
  remove from J the tuple $\arg\min_{t \in J \cap D_?}\{\rho(t)\}$;

We now present our Algorithm 5. Intuitively, it iterates over the tuples in order of proximity to x. For each tuple t_i, it attempts to construct the most promising k-neighborhood that includes tuples from $\{t_1, \ldots, t_i\}$. Each loop in the algorithm can be executed in time $O(1)$. Indeed, we can implement this by keeping three sets with tuples depending on their labels, where we can do insertion and deletion in constant time.

Note that the running time of Algorithm 5 is independent of k, m and $|D_?|$, but it does depend linearly on the number of labels. ◀

Or-Sets. In this uncertain model, each attribute value of a tuple is an *or-set* consisting of finite values (e.g., $\langle 2, 5, 10\rangle$). Each possible world in $\mathcal{I}_{or}$ is formed by choosing exactly one value from each or-set, independent of the choices across all other or-sets. We can express $\mathcal{I}_{or}$ as subset repairs for the following schema: add to R an extra attribute id. Then, assign a unique value for this attribute to each tuple t, and create a tuple for each "realization" of this tuple with the same id. This will increase the input size of the problem, but by at most a polynomial factor (since d is fixed). Moreover, the FD schema is clearly equivalent to an lhs chain; in fact, this is the single primary key case. As a consequence, we have the following proposition.

▶ **Proposition 12.** *We can certify robustness in $\mathcal{I}_{or}$ for k-NNs in P.*

Codd tables. In a *Codd table*, a missing value is represented as Null paired with a domain Dom from which that value can be chosen. A repair is any possible completion of the table. The first observation is that, by adding a new identifier attribute ID, we can think of a Codd table as an inconsistent instance with a single primary key, where each block has a consistent label (i.e., every tuple in the block has the same label). However, if Dom is given as an infinite interval, then Algorithm 2 does not apply directly since it may not be possible to write all tuple completions in an increasing order (there are uncountably many). Indeed, in [13] Karlas et al. consider only the situation where Dom is given as a finite discrete set.

Formally, the distance between a tuple t in the Codd table and a test point x is given by the function $f(x, t) = g_t(y_1, y_2, \ldots, y_n)$ where the y_i's are the missing entries in t. In order to be able to certify robustness, we need to assume that the minimum and maximum of this function can be computed efficiently. This is a mild assumption, since for example for the p-norm the minimum and maximum is achieved when each summand is minimized or maximized respectively.

Now, we observe that since every block has the same label, we can modify Algorithm 2 so that every resulting repair consists of only the first or the last tuple in a block without changing its correctness properties. With this observation in hand, we can now simply define the extremal set $S = \{\min f(x,t) \cup \max f(x,t) \mid t \in D\}$ consisting of the minimum and maximum of each block and then run Algorithm 2 on S. Since S is at most twice the size of S, this implies a linear time algorithm (w.r.t. the number of tuples, labels, and k) for certifiable robustness on Codd tables. We have shown the following proposition.

▶ **Proposition 13.** *We can certify robustness in a Codd table for k-NNs in linear time even if the domain Dom is given as an infinite interval.*

10 Conclusion

In this paper, we study the complexity of certifiable robustness for k-NN classification under FD constraints. We show a dichotomy in the complexity of the problem depending on the structure of the FDs, and we prove that the same dichotomy condition also holds for the counting version of the problem. We envision this work to be a first step towards the long-term goal of investigating the complexity of certifying robustness for other widely used classification algorithms, such as decision trees, Naive Bayes classifiers and linear classifiers.

References

1 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999. doi:10.1145/303976.303983.

2 Jan Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, volume 4353 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2007. doi:10.1007/11965893_1.

3 Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320. PMLR, 2019. URL: http://proceedings.mlr.press/v97/cohen19c.html.

4 Ilias Diakonikolas, Gautam Kamath, Daniel Kane, Jerry Li, Ankur Moitra, and Alistair Stewart. Robust estimators in high-dimensions without the computational intractability. *SIAM J. Comput.*, 48(2):742–864, 2019. doi:10.1137/17M1126680.

5 Samuel Drews, Aws Albarghouthi, and Loris D'Antoni. Proving data-poisoning robustness in decision trees. In *PLDI*, pages 1083–1097. ACM, 2020. doi:10.1145/3385412.3385975.

6 Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004. doi:10.1007/s00453-003-1073-y.

7 Austen Z. Fan and Paraschos Koutris. Certifiable robustness for nearest neighbor classifiers. *CoRR*, abs/2201.04770, 2022. URL: http://arxiv.org/abs/2201.04770.

8 Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007. doi:10.1016/j.jcss.2006.10.013.

9 Leslie Ann Goldberg, Rob Gysel, and John Lapinskas. Approximately counting locally-optimal structures. *J. Comput. Syst. Sci.*, 82(6):1144–1160, 2016. doi:10.1016/j.jcss.2016.04.001.

10 Sergio Greco, Cristian Molinaro, and Francesca Spezzano. *Incomplete Data and Data Dependencies in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012. doi:10.2200/S00435ED1V01Y201207DTM029.

11 Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984. doi:10.1145/1634.1886.

12 Jinyuan Jia, Xiaoyu Cao, Binghui Wang, and Neil Zhenqiang Gong. Certified robustness for top-k predictions against adversarial perturbations via randomized smoothing. In *ICLR*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=BkeWw6VFwr.

13 Bojan Karlas, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, Wentao Wu, and Ce Zhang. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *Proc. VLDB Endow.*, 14(3):255–267, 2020. doi:10.5555/3430915.3442426.

14 Phokion G. Kolaitis and Enela Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012. doi:10.1016/j.ipl.2011.10.018.

15 Paraschos Koutris and Dan Suciu. A dichotomy on the complexity of consistent query answering for atoms with simple keys. In *ICDT*, pages 165–176. OpenProceedings.org, 2014. doi:10.5441/002/icdt.2014.19.

16 Paraschos Koutris and Jef Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In *PODS*, pages 17–29. ACM, 2015. doi:10.1145/2745754.2745769.

17 Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.*, 9(12):948–959, 2016. doi:10.14778/2994509.2994514.

18 Aounon Kumar, Alexander Levine, Tom Goldstein, and Soheil Feizi. Curse of dimensionality on randomized smoothing for certifiable robustness. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 5458–5467. PMLR, 2020. URL: http://proceedings.mlr.press/v119/kumar20b.html.

19 Leonid Libkin. Incomplete information and certain answers in general data models. In *PODS*, pages 59–70. ACM, 2011. doi:10.1145/1989284.1989294.

20 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:10.1145/3196959.3196980.

21 Ester Livshits, Benny Kimelfeld, and Jef Wijsen. Counting subset repairs with functional dependencies. *J. Comput. Syst. Sci.*, 117:154–164, 2021. doi:10.1016/j.jcss.2020.10.001.

22 Simon Razniewski and Werner Nutt. Completeness of queries over incomplete databases. *Proc. VLDB Endow.*, 4(11):749–760, 2011. URL: http://www.vldb.org/pvldb/vol4/p749-razniewski.pdf.

23 Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11):1190–1201, 2017. doi:10.14778/3137628.3137631.

24 Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and J. Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 8230–8241. PMLR, 2020. URL: http://proceedings.mlr.press/v119/rosenfeld20b.html.

25 Anish Das Sarma, Omar Benjelloun, Alon Y. Halevy, and Jennifer Widom. Working models for uncertain data. In *ICDE*, page 7. IEEE Computer Society, 2006. doi:10.1109/ICDE.2006.174.

26 Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *NIPS*, pages 3517–3529, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/9d7311ba459f9e45ed746755a32dcd11-Abstract.html.

27 M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980. doi:10.1137/0138030.

Improved Approximation and Scalability for Fair Max-Min Diversification

Raghavendra Addanki
Manning College of Information & Computer Sciences,
University of Massachusetts Amherst, MA, USA

Andrew McGregor
Manning College of Information & Computer Sciences,
University of Massachusetts Amherst, MA, USA

Alexandra Meliou
Manning College of Information & Computer Sciences,
University of Massachusetts Amherst, MA, USA

Zafeiria Moumoulidou
Manning College of Information & Computer Sciences,
University of Massachusetts Amherst, MA, USA

Abstract

Given an n-point metric space $(\mathcal{X}, d)$ where each point belongs to one of $m = O(1)$ different categories or groups and a set of integers $k_1, \ldots, k_m$, the fair Max-Min diversification problem is to select k_i points belonging to category $i \in [m]$, such that the minimum pairwise distance between selected points is maximized. The problem was introduced by Moumoulidou et al. [ICDT 2021] and is motivated by the need to down-sample large data sets in various applications so that the derived sample achieves a balance over *diversity*, i.e., the minimum distance between a pair of selected points, and *fairness*, i.e., ensuring enough points of each category are included. We prove the following results:

1. We first consider general metric spaces. We present a randomized polynomial time algorithm that returns a factor 2-approximation to the diversity but only satisfies the fairness constraints in expectation. Building upon this result, we present a 6-approximation that is guaranteed to satisfy the fairness constraints up to a factor $1 - \epsilon$ for any constant ϵ. We also present a linear time algorithm returning an $m + 1$ approximation with exact fairness. The best previous result was a $3m - 1$ approximation.
2. We then focus on Euclidean metrics. We first show that the problem can be solved *exactly* in one dimension. For constant dimensions, categories and any constant $\epsilon > 0$, we present a $1 + \epsilon$ approximation algorithm that runs in $O(nk) + 2^{O(k)}$ time where $k = k_1 + \ldots + k_m$. We can improve the running time to $O(nk) + \text{poly}(k)$ at the expense of only picking $(1 - \epsilon)k_i$ points from category $i \in [m]$.

Finally, we present algorithms suitable to processing massive data sets including single-pass data stream algorithms and composable coresets for the distributed processing.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases algorithmic fairness, diversity maximization, data selection, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.7

Related Version *Extended Version*: https://arxiv.org/abs/2201.06678

Funding This work was supported by the NSF under grants CCF-1934846, CCF-1908849, CCF-1637536, CCF-1763423, IIS-1943971, and an Adobe Research Grant.

1 Introduction

Given a universe of n elements $\mathcal{X}$ and a metric distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$, the Max-Min diversification problem seeks to select a k-sized subset $\mathcal{S}$ of $\mathcal{X}$ such that the minimum distance between the points in $\mathcal{S}$ is maximized [23, 48]. Intuitively, the goal is to maximize the *dissimilarity* across all the selected points while k is typically much smaller than n. A considerable amount of work in the database community has addressed the diversity maximization problem in the context of query result diversification [28, 32, 52], efficient indexing schemes for result diversification [6, 29, 54], nearest neighbor search [1], ranking schemes [8, 47], and recommendation systems [2, 15].

Recently, Moumoulidou et al. [46] introduced the *fair* variant of the Max-Min diversification problem. Specifically, the assumption is that the universe of elements $\mathcal{X}$ is partitioned into $m = O(1)$ disjoint categories or groups. Then, the aim is to construct a diverse set of points where each group is sufficiently represented. To this end, the input of the problem includes non-negative integers $k_1, \ldots, k_m$ and the goal now is to select a subset $\mathcal{S}$ using k_i representatives from each group such that the minimum distance across all points is maximized. As a concrete example, consider a query over a maps service for finding restaurants around Manhattan at NYC. Then the goal is to present the user with a diversified set of restaurant locations while representing different cuisines in the sample.

In this work, we improve currently known approximation results for fair Max-Min diversification. This includes improving the approximation factor in the most general case of the problem; significantly decreasing the approximation factor if we slightly relax the fairness constraints; and reducing the approximation factors to arbitrarily close to 1 when the underlying metric is Euclidean. Before presenting our results, we review related work.

1.1 Related Work

The problem of unconstrained diversity maximization, i.e., when the number of groups $m = 1$, is well-studied in the context of facility location, information retrieval, web search and recommendation systems [8, 16, 23, 32, 35, 37, 41, 45, 47, 48, 52]. We refer the interested readers to the following surveys related to the diversification literature [30, 31].

Among popular diversification models are the *distance-based* models. In these models, the diversity of a set of points is modeled via some function defined over pairwise distances. Max-Sum (also known as *remote-clique*) and Max-Min (also known as *remote-edge* or *p-dispersion*) are two of the most well-established distance-based diversification models [42]. In Max-Sum, diversity is defined as the sum of the pairwise distances of points selected in a set, while in Max-Min the diversity of a set is equal to the minimum pairwise distance. For both problems, there are known 2-approximation algorithms, which yield the best approximation guarantee that can be achieved for both problems [12, 15, 48]. There are also recent works on *distance-based* diversity maximization models in the streaming, distributed, and sliding-window models [7, 13, 19, 42].

Contrary to unconstrained diversity maximization, the problem of fair diversity maximization is less studied. To the best of our knowledge, there is a known 2-approximation local search algorithm for fair Max-Sum diversification [2, 14, 15] where fairness is modeled via partition matroids [49]. Recent work also extends the local search approach to distances of negative type [22]. Another recently studied objective called Sum-Min [12] is defined as the sum of distances of all points to their closest point in the set. Bhaskara et al. [12] present an 8-approximation algorithm for Sum-Min under partition matroid constraints.

The most relevant result to our work is due to Moumoulidou et al. [46] that introduced the fair variant for the Max-Min diversification problem that we also study. The proposed fairness objectives have been widely studied by prior work [11, 20, 21, 24, 25, 34, 43, 44, 50, 53, 55, 56, 57], and are based on the definition of group fairness and statistical parity [33]. It is worth noting that there are other definitions for fairness, like individual or causal fairness [36], but these are not the focus of our work. Moumoulidou et al. [46] designed a polynomial time algorithm that achieved a $3m - 1$- approximation for fair Max-Min diversification. There is also a recent line of work for designing (composable) coresets for various distance-based diversification objectives in the fairness setting [17, 18]. Coresets are small subsets of the original data that contain a good approximate solution and are typically used for speed up purposes or designing streaming and distributed algorithms. Prior efforts leave as an open question the construction of coresets for the fair variant of the Max-Min diversification objective.

1.2 Our Results

We present results for both the cases of general metrics and Euclidean metrics.

1. **General Metrics.** In Section 3.1, we present a randomized polynomial time algorithm that returns a factor 2-approximation to the diversity but only satisfies the fairness constraints in expectation, i.e., for each $i \in [m]$, the output is expected to include at least k_i points from $\mathcal{X}_i$. In Section 3.2, we present a 6-approximation that is guaranteed to include $(1-\epsilon)k_i$ points in each group $i \in [m]$ assuming each $k_i = \Omega(\epsilon^{-2} \log m)$. Both these results are based on randomized rounding of a linear program. Finally, in Section 3.3 we present a linear time algorithm returning an $m+1$ approximation with perfect fairness. This is an improvement over the previously known $3m-1$ approximation [46]. We also present an example that shows that the analysis presented in Moumoulidou et al. [46] cannot be improved to obtain a better approximation factor. In Section 3.4, we present a hardness of approximation result arguing that we cannot get an approximation factor better than 2, even allowing for multiplicative approximations in fairness constraints.
2. **Euclidean Metrics.** If the points can be embedded in low dimensional space $\mathbb{R}^D$ (e.g., if the points correspond to geographical locations) and the distances correspond to Euclidean distances then we can significantly improve the approximation factors of our algorithms. In Section 4.1, we show that the problem can be solved *exactly* for $D = 1$. For constant dimensions, groups, we then present a $1+\epsilon$ approximation algorithm that runs in $O(nk) + 2^{O(k)}$ time where $k = k_1 + k_2 + \ldots + k_m$. In Section 4.3, we show how to improve the running time to $O(nk) + \text{poly}(k)$ at the expense of only picking $(1-\epsilon)k_i$ points from group $i \in [m]$. All these results are based on a new coreset construction.

In Sections 5.1 and 5.2, we present algorithms suitable to processing massive data sets including single-pass data stream algorithms and composable coresets for distributed processing.

2 Background and Preliminaries

2.1 Fair Max-Min Diversification

We formally define the problem of fair Max-Min diversification recently introduced in [46].

▶ **Definition 1** (Fair Max-Min). *Let $(\mathcal{X}, d)$ be a metric space where $\mathcal{X} = \bigcup_{i=1}^{m} \mathcal{X}_i$ is a universe of n elements partitioned into m non-overlapping groups and $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a metric distance function. Then $\forall u, v \in \mathcal{X}$, d satisfies the following properties: (1) $d(u, v) = 0$*

iff $u = v$ *(identity), (2)* $d(u,v) = d(v,u)$ *(symmetry), and (3)* $d(u,v) \leq d(u,w) + d(w,v)$ *(triangle inequality). Further, let* $k_1, k_2, \cdots, k_m$ *be non-negative integers with* $k_i \leq |\mathcal{X}_i|$, $\forall i \in [m]$. *The problem of fair Max-Min diversification is now defined as follows:*

$$\begin{aligned} &\underset{\mathcal{S} \subseteq \mathcal{X}}{\text{maximize}} && \min_{\substack{u,v \in \mathcal{S} \\ u \neq v}} d(u,v) \\ &\text{subject to} && |\mathcal{S} \cap \mathcal{X}_i| = k_i, \ \forall i \in [m] \quad \text{(fairness constraints)} \end{aligned}$$

The aim is to select a subset $\mathcal{S} \subseteq \mathcal{X}$ of points that maximizes the minimum pairwise distance across the points in $\mathcal{S}$ while being constrained to include k_i points from group i. Throughout the paper we refer to the diversity of a set $\mathcal{S}$ as $\text{div}(\mathcal{S}) = \min_{u,v \in \mathcal{S}, u \neq v} d(u,v)$.

Let $\mathcal{S}^* = \bigcup_{i=1}^{m} \mathcal{S}_i^*$ be the set of points that obtains the optimal diversity score denoted by $\text{div}(\mathcal{S}^*) = \ell^*$. We say a subset of points $\mathcal{S}$ is an α approximation if $\text{div}(\mathcal{S}) \geq \ell^*/\alpha$ and achieves β fairness if $|\mathcal{S} \cap \mathcal{X}_i| \geq \beta k_i$ for all $i \in [m]$. When $\beta = 1$, we say subset achieves *perfect fairness.*

Fair Max-Min is an NP-hard problem for which the best known polynomial time algorithms are: a 4-approximation algorithm that only works for $m = 2$ groups and a $3m - 1$-approximation algorithm that yields the best guarantees for any $m \geq 3$ [46]. The best approximation factor one can hope for in general metric spaces is a 2-approximation guarantee. This claim easily follows since when $m = 1$, the problem is just the Max-Min diversification problem where it is known that no polynomial time algorithm with an approximation factor better than 2 exists if $P \neq NP$ [48]. We use $\text{poly}(\cdot)$ to describe polynomial time algorithms using the context dependent parameters.

2.2 Low Doubling Dimension Spaces

Our results for low dimensional Euclidean metrics use the fact that such metrics have *low doubling dimension.* Our work in this direction is inspired by work on diversity maximization by Ceccarello et al. [17, 18, 19]. We define a ball of radius r centered at $p \in \mathcal{X}$ as the set of all points in $\mathcal{X}$ within distance strictly less than r from p. We use the notation: $\mathbf{B}(p,r) = \{q \in \mathcal{X} \mid d(p,q) < r\}$.

▶ **Definition 2** (Doubling Dimension)**.** *Let* $(\mathcal{X}, d)$ *be a metric space. The* doubling dimension *of* $\mathcal{X}$ *is the smallest integer* λ *such that any ball* $\mathbf{B}(p,r)$ *of radius* r *around a point* $p \in \mathcal{X}$ *can be covered using at most* $(r/r')^\lambda$ *balls of radius* r'*. The Euclidean metric on* $\mathbb{R}^D$ *has doubling dimension* $O(D)$ *[10, 19, 40].*

2.3 Coresets

Coresets are powerful theoretical tools for designing efficient optimization algorithms in the presence of massive datasets in sequential, streaming or distributed environments [4, 42]. At a high level, coresets are carefully chosen subsets of the original universe of elements that contain an approximate solution to the optimal solution for the optimization problem at hand. A coreset for fair Max-Min diversification is defined as follows:

▶ **Definition 3** (Coreset for Fair Max-Min)**.** *A set* $\mathcal{T} \subseteq \mathcal{X}$ *is an* α*-coreset if there exists a subset* $\mathcal{T}' \subseteq \mathcal{T}$ *with* $|\mathcal{T}' \cap \mathcal{X}_i| = k_i \ \forall i \in [m]$ *and* $\text{div}(\mathcal{T}') \geq \ell^*/\alpha$.

Note that optimally solving Fair Max-Min on $\mathcal{T}$, a set typically much smaller in size than $\mathcal{X}$, yields an α-approximation factor. Further, the notion of coresets is useful for designing algorithms in the distributed setting using the *composability* property. *Composable*

coresets closely relate to the notion of *mergeable summaries* [5, 42] while the assumption is that the universe of elements $\mathcal{X}$ is partitioned into L subsets (e.g., processing sites). Then the goal is to process each subset independently and extract a *local* coreset such that in the union of these local coresets, there is an approximate solution for the optimization problem at hand. Specifically, for Fair Max-Min a composable coreset is defined as follows:

▶ **Definition 4** (Composable coreset for Fair Max-Min). *A function $c(\mathcal{X})$ that maps a set of elements to a subset of these elements computes an $\alpha-$composable coreset for some $\alpha \geq 1$, if for any partitioning*[1] *of $\mathcal{X} = \bigcup_j \mathcal{Y}_j$ and $\mathcal{T} = \bigcup_j c(\mathcal{Y}_j)$, there exists a set $\mathcal{T}' \subseteq \mathcal{T}$ with $|\mathcal{T}' \cap \mathcal{X}_i| = k_i \ \forall i \in [m]$ such that $\text{div}(\mathcal{T}') \geq \ell^*/\alpha$.*

3 General Metrics

In this section, we present algorithms for Fair Max-Min with an arbitrary metric. Our first two algorithms are based on rounding a suitable linear program. In Section 3.3 we present a linear time algorithm returning an $m+1$ approximation with perfect fairness. Finally, in Section 3.4, we give hardness of approximation results for Fair Max-Min.

3.1 2-Approx with Expected Fairness

In this section and others, we assume a guess γ on the optimal diversity value for Fair Max-Min. Note there are at most $\binom{n}{2}$ possible values for the optimal diversity corresponding to the set of distances between pairs of points. Hence, trying all these guesses only increases the running time by a factor $O(n^2)$. Assuming the ratio between the largest and smallest distance is poly(n), this can be reduced to $O(\epsilon^{-1} \log n)$ at the expense of introducing an additional factor of $1+\epsilon$ in the approximation. This follows by the standard technique of only considering guesses that are powers of $(1+\epsilon)$ [39].

Fair Max-Min LP. Let $\mathcal{X} = \{p_1, \ldots, p_n\}$. For every point $p_j \in \mathcal{X}$, we have a variable x_j. We represent the fairness constraint for every group $i \in [m]$ using constraint (1). Additionally, for every point $p \in \mathcal{X}$, we add the constraint (2) that includes at most one point in a ball of radius $\gamma/2$ centered at p. This ensures that the selected points are separated by a distance of at least $\gamma/2$. Using constraint (3), we allow x_p to take any value between 0 and 1. If $\gamma \leq \ell^*$, observe that the optimal solution for Fair Max-Min is a feasible solution for this LP.

$$\sum_{p_j \in \mathcal{X}_i} x_j \geq k_i \quad \forall i \in [m]. \tag{1}$$

$$\sum_{p_\ell \in \mathbf{B}(p,\gamma/2)} x_\ell \leq 1 \quad \forall p \in \mathcal{X}. \tag{2}$$

$$x_j \geq 0 \quad \forall j \in [n]. \tag{3}$$

Let x_j^* denote the optimal solution of the linear program stated above. Let $n' = |\{j : x_j^* > 0\}|$ and without loss of generality suppose $x_j^* > 0$ for all $j \in [n']$. We obtain an integral solution using a randomized rounding algorithm, in which we generate a random ordering based on sampling without replacement, such that a point p_j is selected as the next point in the ordering with probability proportional to x_j^*. This allow us to show (see Lemma 5) that

[1] The notion of composable coresets can also be extended when $\mathcal{X}$ is not divided into disjoint subsets but this is not the focus of our work.

the rounding scheme returns a set $\mathcal{S}$ with at least k_i points *in expectation* from each group $i \in [m]$ (satisfying constraint (1) in expectation). Further, our rounding scheme selects at most one point from each ball of radius $\gamma/2$ (satisfying constraint (2)). Since for a $\gamma \leq \ell^*$ there is a set $\mathcal{S}$ that satisfies the properties discussed above, selecting the set $\mathcal{S}$ for the largest guess γ results in a 2-approximation for the diversity score.

Randomized Rounding. We generate a random ordering σ of $[n']$ where $\sigma(t)$ is randomly chosen from $R_t = [n'] \setminus \{\sigma(1), \ldots, \sigma(t-1)\}$ such that for $j \in R_t$, $\Pr[\sigma(t) = j] = \frac{x^*_j}{\sum_{\ell \in R_t} x^*_\ell}$. After generating the ordering σ, we construct the output set $\mathcal{S}$ by including the point p_j in $\mathcal{S}$ iff $\sigma(j) \leq \sigma(\ell)$ for all $p_\ell \in \mathbf{B}(p_j, \gamma/2)$. Note that all points in the output are at least distance $\gamma/2$ apart.

▶ **Lemma 5.** *There is an algorithm that returns a set $\mathcal{S}$, such that for all groups $i \in [m]$, it holds that $\mathbb{E}[|\mathcal{S} \cap \mathcal{X}_i|] \geq k_i$. Further all the points selected in $\mathcal{S}$ are at least $\gamma/2$ far apart.*

Proof. Consider the randomized rounding algorithm described in this section. Now, let p_j be a point with $x^*_j > 0$. Define A_t to be the event $d(p_{\sigma(t)}, p_j) < \gamma/2$ and $d(p_{\sigma(t')}, p_j) \geq \gamma/2$ for all $t' < t$. In other words, A_t is the event that the first point included in $\mathcal{S}$ from the ball $\mathbf{B}(p_j, \gamma/2)$ is the point from the t-th step (in the ordering σ). Then,

$$\begin{aligned}
\Pr[p_j \in \mathcal{S}] = \sum_{t=1}^{n'} \Pr[\sigma(t) = j | A_t] \Pr[A_t] &= \sum_{t=1}^{n'} \frac{x^*_j}{\sum_{p_\ell \in \mathbf{B}(p_j, \gamma/2)} x^*_\ell} \Pr[A_t] \\
&= \frac{x^*_j}{\sum_{p_\ell \in \mathbf{B}(p_j, \gamma/2)} x^*_\ell} \sum_{t=1}^{n'} \Pr[A_t] \\
&= \frac{x^*_j}{\sum_{p_\ell \in \mathbf{B}(p_j, \gamma/2)} x^*_\ell} \geq x^*_j
\end{aligned}$$

where the last equality follows because $\sum_{t=1}^{n'} \Pr[A_t] = 1$ and the last inequality holds because of constraint (2) in the Fair Max-Min LP. Then for $i \in [m]$, we have $\mathbb{E}[|\mathcal{S} \cap \mathcal{X}_i|] \geq \sum_{p \in \mathcal{X}_i} x^*_p \geq k_i$ where the last inequality follows from constraint (1). ◀

3.2 6-Approx with $(1 - \epsilon)$ Fairness

We now present a more involved rounding scheme of the LP given in the previous section that ensures that the selected points contain at least $(1 - \epsilon)k_i$ points in $\mathcal{X}_i$ for each $i \in [m]$. However, this guarantee comes at the expense of increasing the approximation factor for the diversity score from 2 to 6.

The main idea behind the new rounding scheme stems from the observation that for any $p_i, p_j \in \mathcal{X}$, if $\mathbf{B}(p_i, \gamma/2)$ and $\mathbf{B}(p_j, \gamma/2)$ are disjoint, then, in the previous rounding scheme, the event that p_i is included in the returned solution is independent of the event that p_j is included. This follows because the relative ordering of the elements in $\{\ell : p_\ell \in \mathbf{B}(p_i, \gamma/2)\}$ in σ is independent of the ordering of the elements in $\{\ell : p_\ell \in \mathbf{B}(p_j, \gamma/2)\}$ in σ. This independence will ultimately allow us to use Chernoff bound to argue concentration of the number of elements chosen from each group $\mathcal{X}_j$ $\forall j \in [m]$.

3.2.1 Randomized Rounding with improved fairness guarantees

We solve the LP in Section 3.1 to get a feasible solution $\{x^*_j\}_{j \in [n]}$. Next, we transform $\{x^*_j\}_{j \in [n]}$ into a feasible solution $\{y^*_j\}_{j \in [n]}$ for the following set of constraints, some of which are no longer linear:

$$\sum_{p_j \in \mathcal{X}_i} y_j \geq k_i \quad \forall i \in [m]. \tag{1'}$$

$$\sum_{p_\ell \in \mathbf{B}(p,\gamma/6)} y_\ell \leq 1 \quad \forall p \in \mathcal{X}. \tag{2'}$$

$$y_j \geq 0 \quad \forall j \in [n]. \tag{3'}$$

$$(0 < y_i \text{ and } 0 < y_j) \Rightarrow d(p_i, p_j) \geq \frac{\gamma}{3} \quad \forall p_i, p_j \in \mathcal{X}_\ell, \forall \ell \in [m] \tag{4'}$$

The constraint (2') ensures that at most one point in a ball of radius $\gamma/6$ is selected (instead of $\gamma/2$ used in Section 3.1) and results in an approximation factor of 6. The constraint (4') ensures that points from the same group with non-zero values are separated by at least $\gamma/3$, which is used to argue $(1-\epsilon)$ fairness (see Theorem 7). The transformation of x^* to y^* can be done by redistributing the values as follows:

(a) For each $p_j \in \mathcal{X}$ with $x^*_j > 0$ satisfying $p_j \in \mathcal{X}_i$ and y^*_j value not yet set, we set:

$$y^*_j \leftarrow \left(\sum_{p_\ell \in \mathbf{B}(p_j,\gamma/3) \cap \mathcal{X}_i} x^*_\ell \right) \text{and } y^*_\ell \leftarrow 0 \text{ for all } p_\ell \in \mathbf{B}(p_j, \gamma/3) \cap (\mathcal{X}_i \setminus \{p_j\}).$$

(b) Finally, for all $p_j \in \mathcal{X}$ with $x^*_j = 0$, we set $y^*_j \leftarrow 0$.

Informally, we are just moving weight to p_j from points of the same group (as p_j) that are at a distance strictly less than $\gamma/3$ from p_j.

▶ **Lemma 6.** *$\{y^*_j\}_{j\in[n]}$ satisfies Constraints (1'-4').*

Proof. Observe that $\{y^*_j\}_{j\in[n]}$ satisfies the constraint (4'). If a point $p_j \in \mathcal{X}_i$ satisfies $y^*_j > 0$, then, it means that we set y^*_ℓ to 0 for every $p_l \in \mathbf{B}(p_j, \gamma/3) \cap (\mathcal{X}_i \setminus \{p_j\})$.

Constraint (2') is satisfied because

$$\sum_{p_\ell \in \mathbf{B}(p_j,\gamma/6)} y^*_\ell \leq \sum_{p_\ell \in \mathbf{B}(p_j,\gamma/6+\gamma/3)} x^*_\ell = \sum_{p_\ell \in \mathbf{B}(p_j,\gamma/2)} x^*_\ell \leq 1$$

since $\{x^*_\ell\}_{\ell\in[n]}$ satisfies constraint (2). Constraint (1') is satisfied because $\sum_{p_j\in\mathcal{X}_i} y^*_j = \sum_{p_j\in\mathcal{X}_i} x^*_j$ and Constraint (3') is trivially satisfied. ◀

We next pick a random permutation σ as in the previous Section 3.1, but now using the values $\{y^*_\ell\}_{\ell\in[n]}$. We add p_j to the output $\mathcal{S}$ if $\sigma(j) \leq \sigma(\ell)$ for all p_ℓ such that $d(p_\ell, p_j) < \gamma/6$. Note that all points in $\mathcal{S}$ are therefore at least a distance of $\gamma/6$ apart.

▶ **Theorem 7.** *Asssume $k_i \geq 3\epsilon^{-2}\log(2m)$ for all $i \in [m]$. There is a $\mathrm{poly}(n, k, \delta^{-1})$ time algorithm that returns a subset of points with diversity $\ell^*/6$ and includes $(1-\epsilon)k_i$ points in each group $i \in [m]$ with probability at least $1-\delta$.*

Proof. Let $Y_p = 1$ if the point $p \in \mathcal{X}$ is included in the output $\mathcal{S}$. Fix $i \in [m]$. The proof of Lemma 5 applied to balls of radius $\gamma/6$ rather than balls of radius $\gamma/2$, ensures that for each $i \in [m]$, $\mathbf{E}[\sum_{p\in\mathcal{X}_i} Y_p] \geq k_i$. The fact $\{Y_p\}_{p\in\mathcal{X}_i}$ are fully independent allows us to apply the Chernoff bound and conclude $\Pr[\sum_{p\in\mathcal{X}_i} Y_p \leq (1-\epsilon)k_i] \leq \exp(-\epsilon^2 k_i/3) \leq 1/(2m)$. Hence, by an application of the union bound, we ensure that with probability at least $1/2$, $|\mathcal{S} \cap \mathcal{X}_i| \geq (1-\epsilon)k_i$ for all $i \in [m]$. Repeating the process $\log \delta^{-1}$ times ensures that at least one of the trials succeeds with probability at least $1-\delta$. ◀

Algorithm 1 FAIR-GREEDY-FLOW.

Input: $\mathcal{X} = \bigcup_{i=1}^{m} \mathcal{X}_i$: Universe of available elements.
$k_1, \ldots, k_m \in \mathbb{Z}^+$.
$\gamma \in \mathbb{R}^+$: A guess of the optimum fair diversity.
Output: k_i points in $\mathcal{X}_i$ for $i \in [m]$.

$\mathcal{R} \leftarrow \mathcal{X}$ denote the set of remaining elements.
$\mathcal{C} \leftarrow \emptyset$ denote a collection of subsets of points (called clusters).
while $|\mathcal{R}| > 0$ **(and)** $|\mathcal{C}| \leq km$ **do**
  $D \leftarrow \emptyset$ denote the current cluster, and $D_{\text{col}} \leftarrow \emptyset$ denote the groups of points in cluster D.
  while an element $p \in \mathcal{R} \cap \mathcal{X}_i$ for some $i \in \{1, 2, \cdots, m\} \setminus D_{\text{col}}$. exists **do**
    if $|D| = 0$ (or) $d(p, x) < \frac{\gamma}{m+1}$ for some $x \in D$ **then**
      $D \leftarrow D \cup \{p\}$ and $D_{\text{col}} \leftarrow D_{\text{col}} \cup \{i\}$.
    end if
  end while
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \bigcup_{p \in D} \mathbf{B}(p, \frac{\gamma}{m+1})$.
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{D\}$.
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{X}_i \ \forall i \in [m]$ if $|\{D \mid D \in \mathcal{C} \text{ and } D \cap \mathcal{X}_i \neq \emptyset\}| \geq k$.
end while
▷Construct flow graph :
Let $\mathcal{C} = \{D_1, D_2, \cdots D_t\}$.
Construct directed graph $G = (V, E)$ where

$$\begin{aligned} V &= \{a, u_1, \ldots, u_m, v_1, \ldots, v_t, b\} \\ E &= \{(a, u_i) \text{ with capacity } k_i : i \in [m]\} \\ &\quad \cup \{(v_j, b) \text{ with capacity } 1 : j \in [t]\} \\ &\quad \cup \{(u_i, v_j) \text{ with capacity } 1 : |\mathcal{X}_i \cap D_j| \geq 1\} \end{aligned}$$

Set $\mathcal{S} \leftarrow \emptyset$. Compute maximum a-b flow in G using Ford-Fulkerson algorithm [26].
if flow size $< k = \sum_i k_i$ **then return** $\emptyset$ ▷Abort
else ▷max flow is k
  $\forall (u_i, v_j)$ with flow equal to 1, add the point in D_j with group i to $\mathcal{S}$.
end if
return $\mathcal{S}$.

Note that Theorem 7 requires the k_i values to be sufficiently large, and such conventions have also been used in prior work [12]. For small k_i values, i.e., $k_i = o(\log n)$, the FAIR-GMM algorithm introduced in Moumoulidou et al. [46] obtains a 5-approximation guarantee in polynomial time. Using an additive Chernoff bound, alternatively, we can find at least $k_i - O(\sqrt{k_i \log m})$ points from each group $i \in [m]$, without the requirement of having large k_i's.

3.3 $(m+1)$-Approx with Perfect Fairness

We now describe FAIR-GREEDY-FLOW (Algorithm 1), an $m+1$-approximation algorithm that ensures perfect fairness. This is an improvement over the previously known $3m-1$ approximation [46]. We also present an example that shows that the analysis presented in Moumoulidou et al. [46] cannot be improved to obtain a better approximation factor. The analysis for FAIR-GREEDY-FLOW is presented in the extended version of the paper [3].

Overview of Fair-Greedy-Flow. We assume a guess γ for ℓ^*. The algorithm proceeds by iteratively building clusters of close points of distinct groups. Our main idea is to select one point from each cluster such that the fairness constraints are guaranteed. First, we describe the procedure for building a cluster. Let D denote a cluster initialized with a point of group $i \in [m]$. Among the available points $\mathcal{R}$, we include a point $p \in \mathcal{R}$, if it is within a distance of $\frac{\gamma}{m+1}$ to some point $x \in D$, and no other point of the same group is already present in D.

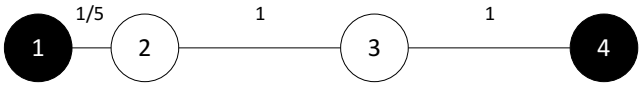
If there is no such point, the cluster D is complete, and we remove all points from $\mathcal{R}$ that are within a distance of $\frac{\gamma}{m+1}$ from some point in D. Also, we discard all points of group i, i.e., $\mathcal{X}_i$ from $\mathcal{R}$, as soon as there are at least k distinct clusters in $\mathcal{C}$ containing points from $\mathcal{X}_i$. We continue this process of iteratively building clusters, until there are points from each group that are part of at least k distinct clusters or if there are no remaining points.

Next, we use an approach similar to [46] and select at most one point from each cluster, satisfying the fairness constraints. We construct a flow network with clusters $D_1, D_2, \cdots, D_t$ in $\mathcal{C}$ represented by nodes $v_1, v_2, \cdots, v_t$ and groups represented by nodes $u_1, u_2, \cdots, u_m$. We add an edge with capacity 1 between every pair u_i and v_j if there is a point of group i in cluster D_j for some $j \in [t]$. We create a source node a and add edges with capacity k_i between a and u_i $\forall i \in [m]$. We then create a sink node b and add edges with capacity 1 between b and v_j $\forall j \in [t]$. Finally, we find maximum flow using Ford-Fulkerson algorithm [26]. For each edge (u_i, v_j) with flow equal to 1, we include the point of group i from cluster D_j in our solution. We conclude with the following theorem:

▶ **Theorem 8.** *Fair-Greedy-Flow Algorithm returns an $(m+1)(1+\epsilon)$-approximation and achieves perfect fairness for the Fair Max-Min problem using a running time of $O(nkm^3\epsilon^{-1}\log n)$.*

We now give a tight example for Fair-Flow in Moumoulidou et al. [46] and show how Fair-Greedy-Flow yields a better approximation.

A tight example for Fair-Flow: a $3m-1$ approximation algorithm [46]. Suppose $k=3$ and we have to select one white and two black points. Here, edges represent the distance across two points, e.g., $d(p_1, p_2) = 1/5$. Note that the optimal solution in this example is the set of points $\{p_1, p_3, p_4\}$ with diversity score equal to 1.



Fair-Flow for a guess $\gamma = 1$, for the black group selects both points since they are at least $d_1 = \frac{m\gamma}{3m-1} = 2/5$ far apart from each other. Similarly for the white group. Now because there is no pair of points with distance strictly less than $d_2 = \frac{\gamma}{3m-1} = 1/5$, Fair-Flow constructs four connected components (each with a point). As a result, the points $\{p_1, p_2, p_4\}$ will be selected by the max-flow algorithm and we obtain a set with diversity score equal to 1/5. Note that for this example, Fair-Greedy-Flow returns the set $\{p_1, p_3, p_4\}$ as p_1 and p_2 are less than 1/3 distance apart. These two points will be in the same cluster and at most one of them can be picked; thus, we guarantee an approximation ratio of 3.

3.4 Hardness of Approximation

In this section, we give a hardness of approximation result for the Fair Max-Min problem. Our result is a generalization and improvement over the 2-approximation hardness shown in [46], as we also allow for approximations in fairness constraints.

▶ **Definition 9** (Gap-Clique$_\rho$). *Given a constant $\rho \geq 1$, a graph G, and an integer k, we want to distinguish between the case where a clique exists of size k (the "yes" case) and the case where no clique exists of size $\geq k/\rho$ (the "no" case).*

It is known that Gap-Clique$_\rho$ is NP-hard for every $\rho \geq 1$ [9]. Now, via a reduction from the Gap-Clique$_\rho$ we argue that Fair Max-Min cannot be approximated to a factor better than 2, even allowing for multiplicative approximations in fairness constraints.

▶ **Theorem 10.** *Let $\alpha < 2$ and $\beta > 0$ be constants. Unless $P = NP$, there is no polynomial time algorithm for the* Fair Max-Min *problem that obtains an α-approximation factor for diversity score, and β fairness.*

Proof. We present a reduction from Gap-Clique$_\rho$, where $\rho = \beta$. For every vertex of the graph G, we create a new point, and set of points is denoted by $\mathcal{X}$. For every edge (u, v) in G, we set $d(u, v) := 2$. For all other pairs of vertices, we set the distances as 1. Every vertex is assigned the same color, and the corresponding fairness constraint is $|\mathcal{S} \cap \mathcal{X}| \geq k$, where $\mathcal{S}$ is the set of points whose diversity we are trying to maximize in Fair Max-Min.

Suppose there is a polynomial time algorithm that returns a set $\mathcal{S}$, obtains an α-approximation for the diversity score, and a β-approximation for the fairness constraints. We first consider the 'Yes' instance in Gap-Clique$_\beta$, i.e., we assume there is a clique of size k in G. This implies $\ell^* = 2$. As $\alpha < 2$, we have that the set $\mathcal{S}$ returned has a diversity score $\geq \ell^*/\alpha > 1$. Therefore, $\mathcal{S}$ is a clique in G as all other pairwise distances are 1 (from construction). As $\mathcal{S}$ is a β-approximation for the fairness constraint, we have that $|\mathcal{S}| \geq k/\beta$. Let us now consider the 'No' instance, i.e., there is no clique of size $\geq k/\beta$ in G. Therefore, $|\mathcal{S} \cap \mathcal{X}| < k/\beta$, as $|\mathcal{S} \cap \mathcal{X}|$ is upper bounded by the maximum clique size in G. From the above arguments, we have that using our algorithm, we can distinguish the 'Yes' and 'No' instances of Gap-Clique$_\beta$, which is not possible unless $P = NP$ [9]. Hence, the theorem. ◀

4 Euclidean Metrics

In this section, we assume that the metric space is Euclidean, i.e., we can associate a point $p_i \in \mathbb{R}^D$ with the ith entry of $\mathcal{X}$ and $d(p_i, p_j) = \|p_i - p_j\|_2 = \sqrt{\sum_{\ell \in [D]} (p_i(\ell) - p_j(\ell))^2}$. When $D = 1$ we show that the problem can be solved exactly in polynomial time via Dynamic Programming. More generally, when $D = O(1)$ we present a bi-criteria approximation that uses an extension of the dynamic programming approach and properties of low dimensional Euclidean spaces.

4.1 Exact Computation in One Dimension

In this section, we assume the points in the universe $\mathcal{X} = \bigcup_{i=1}^{m} \mathcal{X}_i$ can be embedded on a line. Specifically, let $\mathcal{X} = \{p_1, \ldots, p_n\}$ where each $p_i \in \mathbb{R}$ and we order the points such that $p_1 \leq p_2 \leq \ldots \leq p_n$. We further assume a guess γ on the optimal diversity score for Fair Max-Min and design the dynamic programming algorithm Fair-Line (Algorithm 2) that computes an exact solution when $\gamma = \ell^*$. See the previous section for a discussion on guessing γ.

Dynamic Programming. Define the dynamic programming table $H \in \{0, 1\}^{(k_1+1)\times\ldots\times(k_m+1)\times n}$ indexed from 0. An entry $H[k'_1, k'_2, \cdots, k'_m, j] \in \{0, 1\}$ is 1 iff there is a subset $\mathcal{S}'$ of the first j points on the line with diversity γ that contains k'_i points from each group $i \in [m]$. To compute the entries of H, we process the points in their order of appearance on the line.

Note that there is a set $\mathcal{S}'$ with k'_i points from each group i among the first j points if: (1) there is such a set among the first $j - 1$ points, or (2) point j belongs to group i for some $i \in [m]$, and among the first j' points there is a set with $k'_1, \cdots, k'_i - 1, \cdots, k'_m$ points from the corresponding groups where $j' < j$ is the largest value such that $d(p_j, p'_j) \geq \gamma$.

See Fair-Line (Algorithm 2) for the resulting algorithm. For simplicity, the algorithm is written to only determine whether it is possible to pick a subset with diversity γ subject to the required fairness constraints. However, the algorithm can be easily extended to construct

Algorithm 2 FAIR-LINE: An exact algorithm for data on a line.

Input: $\mathcal{X} = \bigcup_{i=1}^{m} \mathcal{X}_i$: Universe of available points.
$k_1, \ldots, k_m \in \mathbb{Z}^+$.
$\gamma \in \mathbb{R}^+$: A guess of the optimum fair diversity.
Output: k_i points in $\mathcal{X}_i$ for $i \in [m]$.

Let $n \leftarrow |\bigcup_{i=1}^{m} \mathcal{X}_i|$ and initialize $H \in \{0,1\}^{(k_1+1)\times\ldots(k_m+1)\times n}$ to 0.
Set $H[0,\ldots,0,0] \leftarrow 1$, $H[0,\ldots,0,1] \leftarrow 1$, and if $p_1 \in \mathcal{X}_\ell$, $H[0,\ldots,\underbrace{1}_{\text{index } \ell},\ldots,0,1] \leftarrow 1$.
for $j = 2$ to n **do**
  Let $i \in [m]$ satisfy $p_j \in \mathcal{X}_i$.
  Let $j' = \max(\{0\} \cup \{j' \in [n] : p_{j'} + \gamma \leq p_j\})$.
  for $k'_1 \in \{0,\ldots,k_1\}, \ldots, k'_m \in \{0,\ldots,k_m\}$ **do**
    $H[k'_1, \cdots, k'_m, j] \leftarrow H[k'_1, \cdots, k'_m, j-1]$.
    If $k'_i \geq 1$, $H[k'_1, \cdots, k'_m, j] \leftarrow H[k'_1, \cdots, k'_i - 1, \ldots, k'_m, j'] \vee H[k'_1, \cdots, k'_m, j-1]$.
  end for
end for
return $H[k_1, k_2, \cdots, k_m, n]$.

a subset of points for every non-zero entry in H by storing a pointer to the choice we made. For an entry $H[k'_1, k'_2, \cdots, k'_m, j] = 1$ that also satisfies $H[k'_1, k'_2, \cdots, k'_m, j-1] = 1$ we store a pointer to that entry. In the second case, if $H[k'_1, k'_2, \cdots, k'_m, j'] = 1$ for some j', we store a pointer to that entry. We construct the solution set using the stored pointers, starting at $H[k_1, k_2, \cdots k_m, n]$ and backtracking, to indicate which points to add to the solution.

▶ **Theorem 11.** *There is an algorithm that solves the FAIR MAX-MIN problem exactly when the points can be embedded on a line and requires a running time of* $O(n^4 \prod_{i=1}^{m}(k_i+1))$.

Proof. We use FAIR-LINE to identify the exact solution. We observe that any optimal solution can be expressed as a subset of the first j points for some $j \in [n]$. From the construction, if the guess $\gamma \leq \ell^*$ there will always be at least k_i points from group i for all $i \in [m]$ that are all γ far apart. Therefore, since the dynamic programming approach finds all the subsets with k_i points per group i for all $j \in [n]$, at least one of the $H[k_1, k_2, \cdots, k_m, j]$ entries will be equal to 1 as required. As discussed previously, we can backtrack and construct the solution set.

Running Time. For a fixed guess γ, we need to compute $\prod_{i=1}^{m}(k_i+1)$ entries for every point, as every k'_i for $i \in [m]$ takes at most $k_i + 1$ values. To compute an entry $H[\cdot, \cdot, \cdots, \cdot, j]$ using FAIR-LINE (Algorithm 2), we need to retrieve $O(n)$ distances to find point j' that is at least γ far apart from point j. Thus, the total running is equal to $O(n^2 \prod_{i=1}^{m}(k_i+1))$ since there are $O(n \prod_{i=1}^{m}(k_i+1))$ entries in H and the computational cost to fill each entry is $O(n)$. As there are $O(n^2)$ distance values the guess γ can take, the total running time is $O(n^4 \prod_{i=1}^{m}(k_i+1))$. ◀

4.2 Coresets for Constant Dimensions

In this section, we design efficient $(1+\epsilon)$-coresets for FAIR MAX-MIN in metric spaces of low doubling dimension (Definition 2). Let λ denote the doubling dimension of $\mathcal{X}$. Our approach generalizes prior work on constructing efficient coresets for unconstrained Max-Min diversification [19] to the FAIR MAX-MIN problem.

Specifically, we give the first algorithm for constructing coresets in metric spaces of doubling dimension. The proposed approach uses the GMM algorithm that obtains a factor 2-approximation for the unconstrained Max-Min diversification problem [48, 51].

GMM is a greedy algorithm and works as follows: it starts with an arbitrary point in a set S and in every subsequent step selects the point that is the farthest away from the previously selected points. In fact, readers familiar with the k-center clustering problem will recognize that this is the same strategy used by [38]. If k is the size of the subset to be selected and n is the size of the universe of points, it is known that GMM can be implemented in $O(kn)$ time [44, 54].

Coreset Construction. First, define $\epsilon' = \epsilon/(1+\epsilon)$ and note that $\epsilon/2 \leq \epsilon' < 1$ since $\epsilon \in (0, 1]$. The Coreset Algorithm constructs coreset $\mathcal{T}$ as follows: we run GMM on each group $i \in [m]$ separately to retrieve a set T_i with $O((4/\epsilon')^{\lambda}k)$ points. The coreset $\mathcal{T}$ is equal to the union of the T_i sets for all $i \in [m]$, namely: $\mathcal{T} \leftarrow \bigcup_{i=1}^{m} T_i$, where $T_i \leftarrow \text{GMM}(\mathcal{X}_i, (4/\epsilon')^{\lambda}k)$.

We will show that $\mathcal{T}$ contains a set $\mathcal{T}^{'}$ with $\text{div}(\mathcal{T}') \geq \ell^*/(1+\epsilon)$ and k_i points from each group i. At a high level, the idea is that for each group i there are two cases: (1) either T_i contains a sufficient number of points that are far apart such that even if we had to remove points close to points selected from other groups, we would still have enough points to satisfy fairness, or (2) the optimal points from group i are within small distance from their closest point in T_i. In the analysis we show that in both cases we have enough points from each group i to satisfy fairness while these points are at least $\ell^*/(1+\epsilon)$ far apart. We first prove the following lemma, which we will use later.

▶ **Lemma 12.** *Let S be a set of $k^{'} = (4/\epsilon')^{\lambda}k$ points that are all at least $(\epsilon'/2)\gamma$ far apart. Then, there exists a subset $S^{'} \subset S$ of points that are all at least γ far apart and $|S^{'}| \geq k$.*

Proof. Let $S' = \emptyset$. Add an arbitrary point x from S to S' and remove all points in the ball $\mathbf{B}(x, \gamma)$ from S. Consider a set of balls of radius $(\epsilon'/4)\gamma$ that cover the removed points. Each of these balls cover at most one removed point since discarded points are at least $(\epsilon'/2)\gamma$ far apart. Hence, the number of balls is at least the number of removed points. But because the doubling dimension is λ we know there exists a set of $(4/\epsilon')^{\lambda}$ balls of radius $(\epsilon'/4)\gamma$ that cover the removed points. Hence, the number of removed points is at most $(4/\epsilon')^{\lambda}$. Since there were $k^{'} = (4/\epsilon')^{\lambda}k$ points in S, we may continue in this way until we've added k points to S'. All chosen points are at least γ apart as required. ◀

Our main theorem in this section is as follows:

▶ **Theorem 13.** *There is an algorithm that returns a $(1+\epsilon)$-coreset of size $O((8/\epsilon)^{\lambda}km)$ in metrics of doubling dimension λ with a running time $O((8/\epsilon)^{\lambda}kmn)$.*

Proof. We show that the set $\bigcup_{i=1}^{m} T_i$ constructed by the Coreset Algorithm is an $(1+\epsilon)$-coreset by showing the existence of a set $\mathcal{T}^{'} \subseteq \bigcup_{i=1}^{m} T_i$ with k_i points from each group i and $\text{div}(\mathcal{T}^{'}) \geq \ell^*/(1+\epsilon)$.

For every group $i \in [m]$, we define $\widehat{T}_i$ to be the maximal prefix of the points added by GMM to form T_i such $\text{div}(\widehat{T}_i) \geq (\epsilon'/2)\ell^*$. We first process all the groups for which $|\widehat{T}_i| < (4/\epsilon')^{\lambda}k$, which we call *critical* groups. For all critical groups, any point $p \in \mathcal{X}_i \setminus \widehat{T}_i$ is within distance $(\epsilon'/2)\ell^*$ from its closest point $f(p)$ in $\widehat{T}_i$, i.e., $d(p, f(p)) < (\epsilon'/2)\ell^*$. As a result, for any pair of optimal points o_1, o_2 in critical groups we deduce:

$$\begin{aligned} d(f(o_1), f(o_2)) &\geq d(o_1, o_2) - d(o_1, f(o_1)) - d(o_2, f(o_2)) \\ &> \ell^* - 2 \cdot \epsilon'\ell^*/2 = \ell^*/(1+\epsilon)\ . \end{aligned}$$

We initialize $\mathcal{T}' = \bigcup_{o \in \cup_{i:\text{critical}} \mathcal{S}_i^*} f(o)$ where $\mathcal{S}_i^*$ is the set of points in an optimal solution belonging to group $\mathcal{X}_i$. We now process all *non-critical* groups $j \in [m]$ in an arbitrary order and remove any point in $\widehat{T}_j$ that is less than ℓ^* apart from some point in $\mathcal{T}'$. Then we argue that in the remaining points there is a set of points T_j' with k_j points that are at least ℓ^* far apart.

By the doubling dimension property and the fact that all the points in $\widehat{T}_j$ are at least $(\epsilon'/2)\ell^*$ far apart, the removal step described above discards at most $(4/\epsilon')^\lambda \sum_{i:\text{processed groups}} |\mathcal{T}' \cap \mathcal{X}_i|$ points from $\widehat{T}_j$. Consequently, regardless of the order in which we process the *non-critical* groups, by the time we process $\widehat{T}_j$ for some $j \in [m]$, there will be *at least* $(4/\epsilon')^\lambda k - \sum_{i:\text{processed groups}} (4/\epsilon')^\lambda k_i \geq (4/\epsilon')^\lambda k_j$ points that are at least $(\epsilon'/2)\ell^*$ apart from each other.

Now by applying Lemma 12 on the points of T_j', we conclude that there are at least k_j points within ℓ^* distance from all other points in $\mathcal{T}'$. Then this set of points T_j' can be added to $\mathcal{T}'$ to satisfy fairness for group j. Thus, it holds that $\text{div}(\mathcal{T}') \geq \ell^*/(1+\epsilon)$ which implies the claimed approximation factor for coreset $\mathcal{T}$. As $\epsilon' = \epsilon/(1+\epsilon) \geq \epsilon/2$, we have $|\mathcal{T}| = O((8/\epsilon)^\lambda km)$. Since we use GMM to obtain $\mathcal{T}$, the running time of the Coreset algorithm is $O((8/\epsilon)^\lambda kmn)$. ◀

From the coreset $\mathcal{T}$, we can obtain a $(1+\epsilon)$-approximation by enumerating over all subsets of $\mathcal{T}$ and returning the subset with maximum diversity and perfect fairness. The running time of this algorithm is $O(2^{O(k)} + nk)$, when m, λ are constants. In the next section, we describe an algorithm that has a polynomial dependence on n and k, obtained at the cost of $(1-\epsilon)$-fairness.

4.3 $(1+\epsilon)$ Approx with $(1-\epsilon)$ Fairness

In this section, we describe Fair-Euclidean (Algorithm 4) which uses $(1+\epsilon)$-coresets described in Section 4.2 and returns a subset of points with diversity at least $\ell^*/(1+\epsilon)$ and has $(1-\epsilon)k_i$ points from each group $i \in [m]$.

First, we discuss Fair-DP (Algorithm 3), which is a dynamic programming subroutine used in Fair-Euclidean. The subroutine will be applied to a collection of t disjoint subsets of $\mathcal{X}$: $\mathcal{C} = \{C_1, C_2, \ldots, C_t\}$. This collection will be *well-separated* in the sense that for all $i \neq j$ and $x \in C_i, y \in C_j$ then $d(x,y) \geq \gamma$. Points in the same set can be arbitrarily close together. We design Fair-DP (Algorithm 3): a dynamic programming algorithm to retrieve a set $\mathcal{F} = \bigcup_{i=1}^m \mathcal{F}_i \subseteq \mathcal{C}$ with k_i points per group i and $\text{div}(\mathcal{F}) \geq \gamma$ if such a set exists in $\mathcal{C}$.

Dynamic Programming. Define the dynamic programming table $H \in \{0,1\}^{(k_1+1)\times\ldots\times(k_m+1)\times t}$ indexed from 0. An entry $H[k_1', k_2', \cdots, k_m', j] \in \{0,1\}$ is 1 iff there is a subset $\mathcal{F}'$ among the first j clusters such that $|\mathcal{F}' \cap \mathcal{X}_i| \geq k_i' \ \forall i \in [m]$ and $\text{div}(\mathcal{F}') \geq \gamma$.

To compute the entries of H, we process the clusters in $\mathcal{C}$ using some fixed ordering. Note that there is a set $\mathcal{F}'$ with k_i' points from each group i among the first j clusters if there is a subset $P \subseteq C_j$ with $\text{div}(P) \geq \gamma$ and p_i' points from each group i; and, among the first $j-1$ clusters, there is a set with $k_1' - p_1', k_2' - p_2', \cdots k_i' - p_i' \cdots, k_m' - p_m'$ points from each group $i \in [m]$ that are at least γ far apart (the function f in Fair-DP (Algorithm 3) evaluates where there is such a set P). We enumerate over all possible subsets of C_j to identify the subset P.

Algorithm 3 FAIR-DP: A dynamic programming subroutine.

Input: $C_1, C_2, \cdots, C_t$: Family of disjoint subsets of $\mathcal{X} = \bigcup_{i=1}^{m} \mathcal{X}_i$.
$k_1, \ldots, k_m \in \mathbb{Z}^+$.
$\gamma \in \mathbb{R}^+$: A guess of the optimum fair diversity.
Output: k_i points in $\mathcal{X}_i$ for $i \in [m]$.

1: Define boolean function $f(p'_1, \ldots, p'_m, j)$ that evaluates to 1 iff there exists $P \subseteq C_j$ with $\text{div}(P) \geq \gamma$ and $|P \cap \mathcal{X}_i| = p'_i$ for all $i \in [n]$.
2: Initialize $H \in \{0,1\}^{(k_1+1)\times\ldots(k_m+1)\times t}$ to 0.
3: Set $H[p'_1, \ldots p'_i, \ldots, p'_m, 1] \leftarrow f(p'_1, \ldots, p'_m, 1)$.
4: **for** $j = 1$ to t **do**
5: For $k'_i \in \{0, \cdots, k_i\}$ $\forall i \in [m]$, update the entries in H as:

$$H[k'_1, \cdots, k'_m, j] \leftarrow \bigvee_{\substack{p'_i \leq k'_i \\ \forall i \in [m]}} H[k'_1 - p'_1, \cdots, k'_m - p'_m, j-1] f(p'_1, \ldots, p'_m, j).$$

6: **end for**
7: **return** $H[k_1, k_2, \cdots, k_m, n]$.

See FAIR-DP (Algorithm 3) for additional details and implementation. For simplicity, the algorithm is written to only determine whether it is possible to pick a subset with diversity γ subject to the required fairness constraints. Similar to FAIR-LINE, the algorithm can be easily extended to construct a subset of points for every non-zero entry in H by storing a pointer to the choice we made.

▶ **Theorem 14.** *If $\gamma = \ell^*$, then, FAIR-DP (Algorithm 3) returns a set $\mathcal{S}$ that satisfies $\text{div}(\mathcal{S}) \geq \ell^*$ and $|\mathcal{S} \cap \mathcal{X}_i| \geq k_i$ $\forall i \in [m]$ and has a running time of $O(\prod_{i=1}^{m}(k_i+1)^2 2^R t)$ where $R = \max\{|C_1|, |C_2|, \cdots, |C_t|\}$.*

Proof. As $\gamma = \ell^*$, the optimal set of points satisfy the fairness constraints. From the construction in FAIR-DP, we will return a set $\mathcal{S}$ that has diversity ℓ^*, and achieves perfect fairness.

Running Time. Consider a value $j \in [t]$. There are $\prod_{i=1}^{m}(k_i+1)$ entries in the table H corresponding to this value of j. For every $k'_i \in \{0, 1, \cdots k_i\}$ and every subset $R \subseteq C_j$ where $|R \cap \mathcal{X}_i| = p'_i$ $\forall i \in [m]$, we check if there is a valid subset of points satisfying fairness constraints using the condition mentioned in FAIR-DP. Since there at most $\prod_{i=1}^{m}(k_i+1)$ ways to enumerate the p'_i values (because $p'_i \leq k'_i$), the total time to compute entries corresponding to this j value is $O(\prod_{i=1}^{m}(k_i+1)^2 2^R)$. Therefore, to compute all the entries in H we need $O(\prod_{i=1}^{m}(k_i+1)^2 2^R t)$ time. ◀

Now, we describe a $1+\epsilon$ approximation algorithm for Euclidean metrics called FAIR-EUCLIDEAN that achieves $1-\epsilon$ fairness.

Overview of Fair-Euclidean. As part of the input, we construct a $(1+\epsilon)$-coreset $\mathcal{T} = \bigcup_{i=1}^{m} T_i$ of size $O((8/\epsilon)^\lambda km)$ using the CORESET algorithm described in Section 4.2. We further assume a guess γ for the optimal diversity score ℓ^*. Note that the coreset $\mathcal{T}$ is only constructed once and used for different guesses of ℓ^*.

For a fixed guess γ, for every group $i \in [m]$, we select a maximal prefix of points $\widehat{T}_i \subset T_i$ that are at least $\epsilon\gamma/4$ far apart and define $\widehat{\mathcal{T}} = \bigcup_{i=1}^{m} \widehat{T}_i$. Our main idea is to partition $\widehat{\mathcal{T}}$ and obtain a collection of sets $\mathcal{C} = \{C_1, C_2, \cdots, C_t\}$ separated by at least γ distance; thus

Algorithm 4 FAIR-EUCLIDEAN: A bi-criteria algorithm.

Input: $\mathcal{X} = \bigcup_{i=1}^{m} \mathcal{X}_i$: points in $\mathbb{R}^D$ with doubling dimension λ.
$k_1, \ldots, k_m \in \mathbb{Z}^+$.
$\mathcal{T} = \bigcup_{i=1}^{m} T_i$: A coreset for fair Max-Min.
$\gamma \in \mathbb{R}^+$: A guess of the optimum fair diversity.
$\epsilon \in [0, 1]$: approximation error parameter.
Output: k_i points in $\mathcal{X}_i$ for $i \in [m]$.
1: $\widehat{T}_i \leftarrow$ a maximal prefix of points in T_i such that $\text{div}(\widehat{T}_i) \geq \epsilon\gamma/4$.
2: $p \leftarrow$ a point selected uniformly at random from $[0, W]^D$, where $W = 2mD\gamma/\epsilon$.
3: Construct axis-aligned cubes $\mathcal{C} = \{C_1, C_2, \cdots, C_t\}$ of side length W using p as one of the corners.
4: In each cube C_i, remove all the points that are within a distance of $\gamma/2$ from one of the boundaries.
5: **return** $\mathcal{S} \leftarrow$ FAIR-DP$(C_1, \cdots, C_t, (1-\epsilon)k_1, \cdots (1-\epsilon)k_m, \gamma)$.

any pair of points $x \in C_i$ and $y \in C_j$, $\forall i, j$ such that $i \neq j$, is separated by distance at least γ. Then, we use FAIR-DP on these sets $C_1, C_2, \cdots, C_t$, and recover a solution $\mathcal{S}$ with diversity γ.

To this end, we partition the points in $\widehat{\mathcal{T}}$ into axis-aligned *cubes* $\mathcal{C} = \{C_1, C_2, \cdots, C_t\}$ of length $W = 2mD\gamma/\epsilon$ as follows: we select a point p uniformly at random from $[0, W]^D$. Using p as one of the corners, we form axis-aligned cubes of length W until every point in $\mathcal{X}$ is in one of the cubes. Then, from every cube C_i $\forall i \in [t]$ we remove every point of $\widehat{T}$ that is within a distance of $\gamma/2$ from one of its boundaries. Notice that any point that was not removed from a cube is at least γ far apart from any other point in a different cube. However, points within the same cube can be arbitrarily close. It is now easy to see that we can use FAIR-DP (Algorithm 3) on $\mathcal{C}$ to retrieve a sufficient number of points from each group in $[m]$. In the analysis below, we show that with probability at least $1/2$, we are able to find at least $(1-\epsilon)k_i$ points from each group $i \in [m]$ that are all γ far apart.

Analysis. Let $\mathcal{S}^* = \bigcup_{i=1}^{m} \mathcal{S}_i^* \subset \mathcal{T}$ denote the optimal solution for FAIR MAX-MIN on the coreset $\mathcal{T} = \bigcup_{i=1}^{m} T_i$ with $\text{div}(\mathcal{S}^*) \geq \ell^*/(1+\epsilon)$. Note that the optimal solution in $\mathcal{T}$ is some subset in $\widehat{\mathcal{T}}$ (see Theorem 13).

As a first step, we bound the number of optimal points $\mathcal{S}_i^*$ from a group $i \in [m]$ that are removed by FAIR-EUCLIDEAN because they are within a distance of $\gamma/2$ from one of the boundaries of a cube.

▶ **Lemma 15.** $\Pr[\forall i \in [m] \ : |\bigcup_{j \in [t]} C_j \cap \mathcal{S}_i^*| \geq (1-\epsilon)k_i] \geq 1/2$.

Proof. Let $T_i' = \bigcup_{j \in [t]} C_j \cap \widehat{T}_i$ be the remaining points in $\widehat{T}_i$ that are not close to the boundaries of any cube. Note that the FAIR-EUCLIDEAN algorithm *succeeds* if after the removal step there are least $(1-\epsilon)k_i$ optimal points from each group i that can be selected by FAIR-DP at the final step of the algorithm while it *fails* otherwise. Below, we show that the probability it *succeeds* is at least $1/2$.

We compute the probability that a point $q \in \widehat{T}_i$ is not removed by FAIR-EUCLIDEAN, i.e., $q \in T_i'$. It is removed if it lies within a distance of $\gamma/2$ from its boundaries in each dimension. Therefore, for q to remain in T_i', the point p selected randomly from $[0, W]^D$ must not fall within a range of total length γ, in each dimension, which gives us:

$$\Pr[q \notin T_i'] = 1 - \Pr[q \in T_i'] = 1 - \left(\frac{W-\gamma}{W}\right)^D \leq \gamma D/W = \epsilon/2m \ .$$

Fix a specific optimum solution. Define A_i be the number of points removed from this solution that are in group i. By Markov's inequality, $\Pr[A_i \geq k_i\epsilon] \leq \frac{\mathbf{E}[A_i]}{k_i\epsilon} \leq \frac{k_i\epsilon/(2m)}{k_i\epsilon} = \frac{1}{2m}$.

Taking union bound over all groups $i \in [m]$, we can bound the probability of discarding more than $k_i\epsilon$ points from some group i, $\Pr[\exists i \in [m] \ : \ A_i \geq k_i\epsilon] \leq \sum_{i=1}^{m} \Pr[A_i \geq k_i\epsilon] < 1/2$, and the lemma follows. ◀

Fair-DP depends exponentially on the number of points remaining in each cube (see Theorem 14). Now, we show that the total number of points remaining in each cube does not depend on n or k, and depends only on m, D, ϵ.

▶ **Lemma 16.** *$|C_j| \leq m \cdot (8mD^{3/2}/\epsilon^2)^{\lambda}$ for all $j \in [t]$.*

Proof. Consider all points in C_j that belong to group i, i.e., $C_j \cap \widehat{T}_i$. From the construction of $\widehat{T}_i \subseteq \mathcal{T}$, we have that every pair of points of the same group is separated by a distance at least $\epsilon\gamma/4$. Therefore, each point can be represented by a ball of radius $\epsilon\gamma/8$, and we want to count the maximum number of non-overlapping balls that can be packed inside the cube C_j. Observe that the length of the diagonal of C_j is $W\sqrt{D}$, and the cube lies entirely in the ball of radius $W\sqrt{D}/2$ with center at the middle of the diagonal. We call this *cube ball*. As Euclidean metrics are doubling metrics, we can cover the *cube ball* with overlapping balls of radius $\epsilon\gamma/8$ and the number of the balls required is $\left(\frac{W\sqrt{D}/2}{\epsilon\gamma/8}\right)^{\lambda}$, where $\lambda = O(D)$ is the doubling dimension of $\mathbb{R}^D$.

We can observe that the total volume occupied by the overlapping balls is at least the volume occupied by the non-overlapping balls corresponding to the points and having the same radius. Therefore, we can upper bound the number of points using the total number of non-overlapping balls used to cover the *cube ball*. As there are m groups, we have that the total number of the points in C_j is: $|C_j| \leq m \cdot \left(\frac{W\sqrt{D}/2}{\epsilon\gamma/8}\right)^{\lambda} = m \cdot \left(8mD^{3/2}/\epsilon^2\right)^{\lambda}$. ◀

We showed that for a fixed guess γ, the *success* probability of Fair-Euclidean is $\geq 1/2$. Note that the only randomization used by Fair-Euclidean is in selecting p. In order to increase the probability of success to $1-\delta$ for some small $\delta \in (0,1)$, we repeatedly select η points uniformly at random from $[0,W]^D$ as the corners. For each corner, we obtain a solution using Fair-Euclidean, and we output the solution with the biggest diversity which also satisfies the fairness constraints with a loss of $(1-\epsilon)$ multiplicative factor. The value of $\eta = \log(1/\delta)$ is selected such that the failure probability is $(1/2)^{\eta} < \delta$.

Note that the construction of the coreset $\mathcal{T}$ allows us to reduce the number of guesses on ℓ^* from $O(n^2)$ to $O(|\mathcal{T}|^2) = O((8/\epsilon)^{2\lambda}k^2m^2)$, which are all the pairwise distances in $\mathcal{T}$. Further, the number of clusters (i.e., cubes) in Fair-Euclidean is upper bounded by the size of the coreset $\mathcal{T}$, which does not depend on n. The running time of Fair-Euclidean depends on the running time to construct the coreset, which is $O((8/\epsilon)^{\lambda}kmn)$, and the running time of Fair-DP (Algorithm 3) on the cubes $\mathcal{C}$. Since the number of points in each cube is $O(m \cdot (8mD^{3/2}/\epsilon^2)^{\lambda})$, we conclude with the following theorem:

▶ **Theorem 17.** *If $\gamma \geq \ell^*/(1+\epsilon)$, Fair-Euclidean Algorithm returns a set $\mathcal{S}$ such that $\mathrm{div}(\mathcal{S}) \geq \ell^*/(1+\epsilon)$ and $|\mathcal{S} \cap \mathcal{X}_i| \geq k_i(1-\epsilon)$ $\forall i \in [m]$ with probability at least $1-\delta$. The running time of the algorithm is:*

$$O(n \cdot (8/\epsilon)^{\lambda}km + \prod_{i=1}^{m}(k_i+1)2^{m(8mD^{3/2}/\epsilon^2)^{\lambda}}(8/\epsilon)^{\lambda}km\log|\mathcal{T}|\log(1/\delta)).$$

Proof. The running time of Fair-Euclidean (Algorithm 4) depends on: (1) the running time of constructing the coreset $\mathcal{T}$ which is $O(nkm(8/\epsilon)^{\lambda})$, where λ is the doubling dimension, and (2) the running time of Fair-DP (Algorithm 3) on the clusters for every guess γ.

From Theorem 14, we know that Fair-DP has a running time of $O(\prod_{i=1}^{m}(k_i+1)^2 2^R t)$, where t is the number of clusters and R is the maximum size across all t clusters. We upper bound the number of clusters by the coreset size. So, $t = O((8/\epsilon)^{\lambda}km)$. From Lemma 16, we have $R = O(m(8mD^{3/2}/\epsilon^2)^{\lambda})$. Combining all the above, the final running time is:

$$O((8/\epsilon)^{\lambda}kmn + \log|\mathcal{T}|\log(1/\delta)\prod_{i=1}^{m}(k_i+1)^2 2^{m(8mD^{3/2}/\epsilon^2)^{\lambda}}(8/\epsilon)^{\lambda}km). \quad \blacktriangleleft$$

We can observe that the running time depends doubly exponentially on the doubling dimension, which is not uncommon for diversity maximization in doubling dimension metrics [17, 19].

5 Scalable Implementations

5.1 Data Stream Algorithms

In this section, we present single pass data stream algorithms that obtain the same approximation guarantees as that of sequential algorithms, while using low space. Missing details from this section are presented in the extended version of the paper [3].

5.1.1 Extending Previous Algorithms

First, we describe an algorithm called τ-GMM that processes points sequentially, and includes a point in the solution if it is at least the threshold τ apart from every point in the current solution set. The set of points returned by τ-GMM are all separated by a distance of at least τ. If $m = 1$, then, we can set $\tau = \ell^*/2$ (using guessing for ℓ^*), and τ-GMM returns a solution set that is also a 2-approximation for the Fair Max-Min problem [27]. τ-GMM allows us to extend it to data streaming setting, unlike the GMM algorithm which requires identifying the maximum distance point in each iteration.

Using τ-GMM with $\tau = \ell^*/2$, we can obtain a 5-coreset for general metrics [46], and $(1+\epsilon)$-coreset for Euclidean metrics (Section 4.2). Then, on the coreset, we use the randomized rounding algorithm from Section 3.2 and return the solution. This approach gives us the following guarantees:

▶ **Corollary 18.** *There is a $O(\epsilon^{-1}km\log n)$-space data stream algorithm that returns a $30(1+\epsilon)$-approximation with $(1-\epsilon)$-fairness for general metrics. For Euclidean metrics, there is a $O((8/\epsilon)^{\lambda}km\epsilon^{-1}\log n)$ space data stream algorithm that returns a $1+\epsilon$-approximation with $(1-\epsilon)$-fairness where λ is the doubling dimension of $\mathcal{X} \subset \mathbb{R}^D$.*

5.1.2 Improved Result for $m = 2$

In [46], the authors describe an algorithm called Fair-Swap which returns a 4-approximation to the Fair Max-Min problem when the number of groups is $m = 2$. The algorithm can be directly extended to a 2-pass streaming algorithm using $O(k)$ space with the same 4-approximation guarantee. Building upon their work, and using new ideas we obtain a *single pass* algorithm Fair-Stream-2Groups which uses $O(k)$ space, and obtains 4-approximation to the Fair Max-Min problem.

The algorithm maintains 3 sets S, S_1, S_2 using τ-GMM for all of them. In S, we include points in a group-agnostic way (similar to Fair-Swap) ignoring the fairness constraints. In S_1, we include points only of group 1, and in S_2, we include points only of group 2. By setting $\tau = \ell^*/2$ we maintain the sets S, S_1 and S_2 such that all points are at least $\ell^*/2$ distance apart in every one of them.

Without loss of generality, suppose $\mathcal{X}_1$ satisfies $|S \cap \mathcal{X}_1| < k_1$. Our algorithm proceeds by identifying $k_1 - |S \cap \mathcal{X}_1|$ additional points from S_1 denoted by Z_1 by running τ-GMM with $\tau = \ell^*/4$. This ensures that the final set of points from group 1, i.e., $(S \cap \mathcal{X}_1) \cup Z_1$ are $\ell^*/4$ apart. By discarding the nearest neighbors of newly added points (i.e., Z_1), in $S \cap \mathcal{X}_2$, we argue that our algorithm obtains a 4-approximation. We obtain the following guarantees:

▶ **Theorem 19.** *There is a one-pass streaming algorithm that returns a $4(1+\epsilon)$-approximation for Fair Max-Min problem using $O(k\epsilon^{-1} \log n)$ space.*

5.2 Composable Coresets

In this section, we design *composable* coresets for Fair Max-Min. We assume the points $\mathcal{X}$ are partitioned into L disjoint sets. We discuss an algorithm for constructing $(1+\epsilon)$-composable coresets for Euclidean metrics, and discuss extensions. Missing details are presented in the extended version of the paper [3].

5.2.1 Constructing $(1+\epsilon)$-composable coresets

We assume the universe of points $\mathcal{X}$ is partitioned into a collection of L disjoint sets $\mathcal{Y}_1, \mathcal{Y}_2, \cdots, \mathcal{Y}_L$. As in Section 4.2, we define an $\epsilon' > 0$ value such that $(1-\epsilon') = 1/(1+\epsilon)$. We generalize the approach for constructing the coreset $\mathcal{T}$ as follows: let $\mathcal{Y}_j^i$ denote the points of group i present in $\mathcal{Y}_j$ for $i \in [m]$ and $j \in [L]$. Then on each partition j and group i, we run GMM to retrieve a diverse set T_i^j with $O((4/\epsilon')^\lambda k)$, or equivalently $O((8/\epsilon)^\lambda k)$ points since $\epsilon' \geq \epsilon/2$. The coreset $\mathcal{T}$ is defined as:

1. For $j \in [L]$, construct T_j: $T_j \leftarrow \bigcup_{i=1}^m T_j^i$, where $T_j^i \leftarrow \text{GMM}(\mathcal{Y}_j^i, (4/\epsilon')^\lambda k)$
2. $\mathcal{T} \leftarrow \bigcup_{j=1}^{L} T_j$

We obtain the following theorem:

▶ **Theorem 20.** *$\mathcal{T}$ is a $(1+\epsilon)$-composable coreset for fair Max-Min diversification of size $O((8/\epsilon)^\lambda kmL)$ in metrics of doubling dimension λ that can be obtained in $O((8/\epsilon)^\lambda kmnL)$ time.*

For general metrics, using a similar approach, we obtain a 5-composable coreset by extending a recent construction of 5-coreset for the sequential setting [46]. In the extended version of the paper, we also discuss two-pass distributed algorithms for constructing α-composable coresets for Euclidean ($\alpha = 1+\epsilon$) and general metrics ($\alpha = 5$).

6 Conclusion

In this paper, we presented new approximation algorithms that substantially improve upon currently known results for the Fair Max-Min problem both in general and Euclidean metric spaces. There are several interesting directions for future work, including obtaining a 2-approximation for the problem in general metrics or improving the hardness result.

Another direction is to generalize the fairness constraints to arbitrary matroid constraints (the fairness constraints considered in this paper can be expressed via the special case of a partition matroid). While there are results known for related diversity maximization problems under matroid constraints [2, 12, 15], to the best of our knowledge, there are currently no results for Max-Min diversification.

References

1 Sofiane Abbar, Sihem Amer-Yahia, Piotr Indyk, Sepideh Mahabadi, and Kasturi R Varadarajan. Diverse near neighbor problem. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, pages 207–214, 2013.

2 Zeinab Abbassi, Vahab S. Mirrokni, and Mayur Thakur. Diversity maximization under matroid constraints. In *KDD '13*, pages 32–40, 2013.

3 Raghavendra Addanki, Andrew McGregor, Alexandra Meliou, and Zafeiria Moumoulidou. Improved approximation and scalability for fair max-min diversification, 2022. `arXiv:2201.06678`.

4 P. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets, 2007.

5 Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. In *PODS '12*, pages 23–34, 2012.

6 Pankaj K. Agarwal, Stavros Sintos, and Alex Steiger. Efficient indexes for diverse top-k range queries. In *PODS '20*, pages 213–227, 2020.

7 Sepideh Aghamolaei, Majid Farhadi, and Hamid Zarrabi-Zadeh. Diversity maximization via composable coresets. In *CCCG*, 2015.

8 Albert Angel and Nick Koudas. Efficient diversity-aware search. In *SIGMOD '11*, pages 781–792, 2011.

9 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998.

10 Patrice Assouad. Plongements lipschitziens dans $\mathbb{R}^n$. *Bulletin de la Société Mathématique de France*, 111:429–448, 1983.

11 Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. *Advances in Neural Information Processing Systems*, 32:4954–4965, 2019.

12 Aditya Bhaskara, Mehrdad Ghadiri, Vahab Mirrokni, and Ola Svensson. Linear relaxations for finding diverse elements in metric spaces. In *NIPS'16*, pages 4105–4113, 2016.

13 Michele Borassi, Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. Better sliding window algorithms to maximize subadditive and diversity objectives. In *PODS '19*, pages 254–268, 2019.

14 Allan Borodin, Aadhar Jain, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions, and dynamic updates. *ACM Trans. Algorithms*, 2017.

15 Allan Borodin, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS '12*, pages 155–166, 2012.

16 Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98*, pages 335–336, 1998.

17 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Fast coreset-based diversity maximization under matroid constraints. In *WSDM '18*, pages 81–89, 2018.

18 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. A general coreset-based approach to diversity maximization under matroid constraints. *ACM Trans. Knowl. Discov. Data*, 2020.

19 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.*, pages 469–480, 2017.

20 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.

21 L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi. Ranking with fairness constraints. In *ICALP*, 2017.

22 Alfonso Cevallos, Friedrich Eisenbrand, and Rico Zenklusen. Local search for max-sum diversification. In *SODA '17*, pages 130–142, 2017.

23 Barun Chandra and Magnús M Halldórsson. Approximation algorithms for dispersion problems. *J. Algorithms*, pages 438–465, 2001.

24 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *NIPS*, 2017.

25 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.

26 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

27 Graham Cormode, S. Muthukrishnan, and Wei Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *ICDE 2007*, pages 1036–1045, 2007. doi:10.1109/ICDE.2007.368962.

28 Ting Deng and Wenfei Fan. On the complexity of query result diversification. *ACM Transactions on Database Systems (TODS)*, 39(2):1–46, 2014.

29 M. Drosou and E. Pitoura. Diverse set selection over dynamic data. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1102–1116, 2014.

30 Marina Drosou, H.V. Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. Diversity in big data: A review. *Big Data*, 5:73–84, 2017.

31 Marina Drosou and Evaggelia Pitoura. Search result diversification. *SIGMOD Rec.*, 39(1):41–47, 2010.

32 Marina Drosou and Evaggelia Pitoura. Disc diversity: Result diversification based on dissimilarity and coverage. *Proc. VLDB Endow.*, 6(1):13–24, November 2012.

33 Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *ITCS '12*, pages 214–226, 2012.

34 Marwa El Halabi, Slobodan Mitrović, Ashkan Norouzi-Fard, Jakab Tardos, and Jakub M Tarnawski. Fairness in streaming submodular maximization: Algorithms and hardness. In *NeurIPS 2020*, volume 33, pages 13609–13622, 2020.

35 Erhan Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.

36 Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: Testing software for discrimination. In *ESEC/FSE '17*, pages 498–510, 2017.

37 Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW '09*, pages 381–390, 2009.

38 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

39 Sudipto Guha. Tight results for clustering and summarizing data streams. In *ICDT '09*, pages 268–275, 2009.

40 Anupam Gupta, Robert Krauthgamer, and James R Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 534–543. IEEE, 2003.

41 Refael Hassin, Shlomi Rubinstein, and Arie Tamir. Approximation algorithms for maximum dispersion. *Oper. Res. Lett.*, 21(3):133–137, October 1997.

42 Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *PODS '14*, pages 100–108, 2014.

43 Matthew Jones, Huy Nguyen, and Thy Nguyen. Fair k-centers via maximum matching. In *ICML 2020*, pages 4940–4949, 2020.

44 Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *ICML '19*, volume 97, pages 3448–3457, June 2019.

45 Michael J. Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987.

46 Zafeiria Moumoulidou, Andrew McGregor, and Alexandra Meliou. Diverse Data Selection under Fairness Constraints. In *ICDT 2021*, pages 13:1–13:25, 2021.

47 Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Diversifying top-k results. *Proc. VLDB Endow.*, 5(11):1124–1135, July 2012.

48 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.

49 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

50 Julia Stoyanovich, Ke Yang, and H. V. Jagadish. Online set selection with fairness and diversity constraints. In *EDBT*, 2018.

51 Arie Tamir. Obnoxious facility location on graphs. *SIAM J. Discrete Math.*, 4:550–567, November 1991.

52 Marcos R. Vieira, Humberto L. Razente, Maria C. N. Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina, and Vassilis J. Tsotras. On query result diversification. In *ICDE 2011*, pages 1163–1174, 2011.

53 Yanhao Wang, Francesco Fabbri, and Michael Mathioudakis. Fair and representative subset selection from data streams. In *WWW 2021*, pages 1340–1350, 2021.

54 Yue Wang, Alexandra Meliou, and Gerome Miklau. Rc-index: Diversifying answers to range queries. *Proc. VLDB Endow.*, 11(7):773–786, 2018.

55 Ke Yang, Vasilis Gkatzelis, and Julia Stoyanovich. Balanced ranking with diversity constraints. In *IJCAI'19*, pages 6035–6042, 2019.

56 Ke Yang and Julia Stoyanovich. Measuring fairness in ranked outputs. In *SSDBM '17*, 2017.

57 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.

Rewriting with Acyclic Queries: Mind Your Head

Gaetano Geck
TU Dortmund University, Germany

Jens Keppeler
TU Dortmund University, Germany

Thomas Schwentick
TU Dortmund University, Germany

Christopher Spinrath
TU Dortmund University, Germany

Abstract

The paper studies the *rewriting problem*, that is, the decision problem whether, for a given conjunctive query Q and a set $\mathcal{V}$ of views, there is a conjunctive query Q' over $\mathcal{V}$ that is equivalent to Q, for cases where the query, the views, and/or the desired rewriting are acyclic or even more restricted.

It shows that, if Q itself is acyclic, an acyclic rewriting exists if there is any rewriting. An analogous statement also holds for free-connex acyclic, hierarchical, and q-hierarchical queries.

Regarding the complexity of the rewriting problem, the paper identifies a border between tractable and (presumably) intractable variants of the rewriting problem: for schemas of bounded arity, the *acyclic rewriting problem* is NP-hard, even if both Q and the views in $\mathcal{V}$ are acyclic or hierarchical. However, it becomes tractable, if the views are free-connex acyclic (i.e., in a nutshell, their body is (i) acyclic and (ii) remains acyclic if their head is added as an additional atom).

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

Keywords and phrases rewriting, acyclic rewriting, acyclic conjunctive queries, free-connex queries, hierarchical queries, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.8

Related Version *Full Version*: `https://arxiv.org/abs/2201.05129` [15]

1 Introduction

The *query rewriting problem* asks, for a query Q and a set $\mathcal{V}$ of views, whether there is a query Q' over $\mathcal{V}$ that is equivalent to Q, and to find such a *rewriting* Q'.

We emphasise that in the literature, various notions of rewriting are studied and that the one above is sometimes called *exact* [8] or *equivalent* [2]. Since we are in this paper exclusively interested in exact rewritings, we use the term rewriting in the sense stated above.

Over the last decades, query rewriting has been studied in a wide range of settings with varying database models (relational, XML, graph, ...), query languages, semantics (set, bag, ...) and more. A particular well-known setting is formed by conjunctive queries (CQ) under set semantics, which captures some core aspects of SQL queries. This is also the setting studied in this paper. We refer to [2, 11, 18] for an overview of the extensive literature.

In this paper we are interested in a modified version of the query rewriting problem which does not merely ask for *any* rewriting but rather for rewritings that are structurally simple and allow for efficient evaluation. This is relevant in a setting where access to the database is only possible through the views, in which case the use of an efficient rewriting is obvious. It can also be interesting in a scenario where multiple queries have to be evaluated and it could be useful to evaluate them in some order that allows many of them to be evaluated efficiently.

Table 1 Complexity results for the acyclic rewriting problem. Note that QHCQ ⊆ CCQ ⊆ ACQ and QHCQ ⊆ HCQ ⊆ ACQ hold. The head arity of a view V is the arity of its head atom head(V). The weak head arity of a view is defined in Definition 6.3. In a nutshell, a view with weak head arity ℓ can be "split" into (multiple) views with head arity at most ℓ.

<table>
<tr><th>$\mathbb{V}$
Views</th><th>$\mathbb{Q}$
Query</th><th>$\mathbb{R}$
Rewriting</th><th>Restriction
of views</th><th>$\textsc{Rewr}^k(\mathbb{V},\mathbb{Q},\mathbb{R})$ for every $k \in \mathbb{N}$
(bounded arity db-schema)</th><th>$\textsc{Rewr}(\mathbb{V},\mathbb{Q},\mathbb{R})$
(unbounded arity db-schema)</th></tr>
<tr><td rowspan="3">ACQ</td><td rowspan="3">ACQ</td><td rowspan="3">ACQ</td><td></td><td colspan="2">NP-complete for $k \geq 3$ (Theorem 5.7)</td></tr>
<tr><td>head arity $\leq \ell$
$\ell \in \mathbb{N}$</td><td colspan="2">in polynomial time (Corollary 5.5)</td></tr>
<tr><td>weak head arity $\leq \ell$
$\ell \in \mathbb{N}$</td><td colspan="2">in polynomial time (Theorem 6.9)</td></tr>
<tr><td>CCQ</td><td rowspan="4"></td><td rowspan="4"></td><td>in polynomial time (Theorem 6.2)</td><td>open</td></tr>
<tr><td>HCQ</td><td rowspan="2">HCQ</td><td rowspan="2">HCQ</td><td colspan="2">NP-complete for $k \geq 3$ (Corollary 7.5)</td></tr>
<tr><td rowspan="2">QHCQ</td><td rowspan="2">in polynomial time
(Corollary 7.4)</td><td rowspan="2">open</td></tr>
<tr><td>QHCQ</td><td>QHCQ</td></tr>
</table>

The forms of structural simplicity that we consider in this paper are acyclic (ACQ) and hierarchical queries (HCQ), and their slightly stronger versions free-connex acyclic (CCQ) and q-hierarchical queries (QHCQ). For a brief discussion of the origin and applications of these classes, we refer to Section 8.

We are interested in two kinds of questions.

(1) Under which circumstances is it guaranteed that a structurally simple rewriting exists, if there exists a rewriting at all?

(2) What is the complexity to decide whether such a rewriting exists and to compute one?

We study these questions depending on the structure of the given views and the given query and we consider the same simplicity notions.

In particular, we study the decision problem $\textsc{Rewr}(\mathbb{V},\mathbb{Q},\mathbb{R})$ that asks whether for a given set of views from $\mathbb{V}$ and a query from $\mathbb{Q}$, there is a rewriting from $\mathbb{R}$, for various classes $\mathbb{V}$, $\mathbb{Q}$ and $\mathbb{R}$, with an emphasis on the case where $\mathbb{R}$ is ACQ or a subclass. In the following we refer to this case as the *acyclic rewriting problem*. To the best of our knowledge, there has been no previous work dedicated to the study of questions (1) and (2).

The answer to question (1) turns out to be very simple and also quite encouraging: in the case that $\mathbb{Q} = \mathsf{ACQ}$, whenever a rewriting exists, there is also an acyclic rewriting. And the same is true for the three subclasses of ACQ. That is, for every query in CCQ, HCQ, or QHCQ, whenever a rewriting exists, there is also a rewriting in CCQ, HCQ, or QHCQ, respectively. Thus, in these cases a rewriting with efficient evaluation exists, if there is a rewriting at all. This answer to question (1) simplifies the study of question (2), since it implies that for $\mathbb{Q} \in \{\mathsf{ACQ}, \mathsf{CCQ}, \mathsf{HCQ}, \mathsf{QHCQ}\}$ and $\mathbb{V} \subseteq \mathsf{CQ}$ the decision problems $\textsc{Rewr}(\mathbb{V},\mathbb{Q},\mathsf{CQ})$ and $\textsc{Rewr}(\mathbb{V},\mathbb{Q},\mathbb{Q})$ – and, thus, their complexities – coincide.

The study of question (2) reveals that the complexity of the acyclic rewriting problem, may depend on two parameters: the arity of the underlying schema and the arity of the views. We denote the restriction of $\textsc{Rewr}(\mathbb{V},\mathbb{Q},\mathbb{R})$ to database schemas of arity at most k by $\textsc{Rewr}^k(\mathbb{V},\mathbb{Q},\mathbb{R})$, and we indicate by $\mathbb{V}^k$ if the arity of views is at most k.

Our main findings regarding the complexity of the acyclic rewriting problem are as follows (see Table 1 for an overview).

- If the query and the views are acyclic and the arity of the views is bounded by some fixed k, then the acyclic rewriting problem is tractable, that is, for every k: $\textsc{Rewr}(\mathsf{ACQ}^k,\mathsf{ACQ},\mathsf{ACQ})$ is in polynomial time (Corollary 5.5). Furthermore, an acyclic

rewriting can be computed in polynomial time (if it exists). This follows easily with the help of the canonical rewriting approach (see [26, Proposition 5.1]) and our answer to question (1).
- If the query and the views are acyclic and the arity of the views can be unbounded, then the acyclic rewriting problem is intractable, even if the database schema has bounded arity, more precisely: $\textsc{Rewr}^3(\mathsf{ACQ}, \mathsf{ACQ}, \mathsf{ACQ})$ is NP-complete (Theorem 5.7).
- If the query is acyclic, the views are free-connex acyclic, and the arity of the database schema is bounded by some fixed k, then the acyclic rewriting problem is tractable, that is, for every k: $\textsc{Rewr}^k(\mathsf{CCQ}, \mathsf{ACQ}, \mathsf{ACQ})$ is in polynomial time (Theorem 6.2).

Many results use a characterisation of rewritability, based on the notion of *cover partitions*, that might be interesting in its own right. Similar notions have been used in earlier work, but ours is particularly suited for the study of exact (equivalent) rewritings.

Our paper is organised as follows: we introduce general basic notions in Section 2 and notions related to rewritings in Section 3. The characterisation of rewritability in terms of cover partitions is given in Section 4. The answer to question (1) and the complexity of the acyclic rewriting problem for acyclic queries and views are studied in Section 5. The feasibility for free-connex acyclic views (and an additional class) is presented in Section 6. Section 7 shows that the answer to question (1) is the same for hierarchical and q-hierarchical queries and views. Section 8 discusses related work and, in particular, the relationship of our characterisation of rewritability with results in the literature. Section 9 concludes.

2 Preliminaries

In this section, we fix some notation and recall the basic concepts from relational databases that are relevant for this paper. Let dom and var be countably infinite sets of *data values* (also called *constants*) and *variables*, respectively. We use the natural extensions of mappings of variables onto tuples, sets without notational distinction. By id we denote the identity mapping (on any set of variables).

Databases and queries are formulated over *schemas* that represent relation names. A *schema* $\mathcal{S}$ is a finite set of relation symbols, where each symbol R is equipped with a fixed arity $\mathrm{ar}(R) \in \mathbb{N}_0$. A *fact* or *R-fact* $R(a_1, \ldots, a_r)$ comprises a relation symbol R with some arity r and data values $a_1, \ldots, a_r$. A *database* D over schema $\mathcal{S}$ is a finite subset of R-facts for $R \in \mathcal{S}$.

Queries. In this paper, we consider conjunctive queries and restrictions of them. These queries are conjunctions of relation atoms. An *atom* is of the form $R(x_1, \ldots, x_r)$ with a relation symbol R with arity r, and variable set $\mathrm{vars}(A) = \{x_1, \ldots, x_r\}$.

We represent a *conjunctive query* (*CQ* for short) over schema $\mathcal{S}$ as a rule of the form $A \leftarrow A_1, \ldots, A_m$ whose *body* $\{A_1, \ldots, A_m\}$ consists of a positive number of atoms and whose *head* is a single atom A such that the following two conditions are satisfied. First, atoms $A_1, \ldots, A_m$ refer to relation symbols from $\mathcal{S}$ and atom A, on the contrary, does not. Second, the query is *safe*, that is, every variable in the head occurs in at least one atom of the body. Let $\mathrm{head}(Q)$ and $\mathrm{body}(Q)$ denote the head and body of a query Q, respectively. Variables that occur in the head of a query are called *head variables*; all other variables are called *quantified variables*. The *arity* of a query Q is the arity of its head atom. Furthermore, a query Q is a Boolean query if the arity of $\mathrm{head}(Q)$ is 0.

A *valuation* is a mapping $\vartheta : \mathsf{var} \rightharpoonup \mathsf{dom}$. A database D *satisfies* a set $\mathcal{A}$ of atoms under a valuation ϑ, if $\vartheta(\mathcal{A}) \subseteq D$, that is for every atom $R(x_1, \ldots, x_r)$ in $\mathcal{A}$ the fact $R(\vartheta(x_1), \ldots, \vartheta(x_r))$ is contained in D. The *result* of query Q on database D is defined as

$$Q(D) = \{\vartheta(\mathrm{head}(Q)) \mid D \text{ satifies } \mathrm{body}(Q) \text{ under } \vartheta\}.$$

Relationships between queries. Queries over the same schema can be compared with respect to the results they define. Let Q_1 and Q_2 be queries. We say that Q_1 is *contained* in Q_2 (notation: $Q_1 \sqsubseteq Q_2$) if $Q_1(D) \subseteq Q_2(D)$ for every database D. We say that Q_1 and Q_2 are *equivalent* (notation: $Q_1 \equiv Q_2$) if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$.

It is well-known that a conjunctive query Q_1 is contained in a conjunctive query Q_2 if and only if there is a homomorphism from the latter query into the first [9]. Such a *homomorphism* is a mapping $h : \text{vars}(Q_2) \to \text{vars}(Q_1)$ such that (1) $h(\text{body}(Q_2)) \subseteq \text{body}(Q_1)$ and (2) $h(\text{head}(Q_2)) = \text{head}(Q_1)$. We call h a *body homomorphism* if it fulfils condition (1).

A conjunctive query Q_1 is *minimal* if there is no conjunctive query Q_2 such that $Q_2 \equiv Q_1$ and $|\text{body}(Q_2)| < |\text{body}(Q_1)|$ holds.

Structurally simple queries. As mentioned in the introduction, we are particularly interested in the following classes of structurally simple queries. A *join tree* for a query Q is a tree whose vertices are the atoms in the query's body that satisfies the following path property: for two atoms $A, A' \in \text{body}(Q)$ with a common variable x, all atoms on the path from A to A' contain x. A query Q is *acyclic* if it has a join tree. It is *free-connex acyclic* if Q is acyclic and the Boolean query whose body is $\text{body}(Q) \cup \{\text{head}(Q)\}$ is acyclic as well [3, 7]. For a fixed query Q and some variable x in Q, let $\text{atoms}(x)$ denote the set of atoms in $\text{body}(Q)$ in which x appears.

▶ **Definition 2.1** ([12], [6, Definition 3.1])**.** A conjunctive query Q is *hierarchical* if, for all variables x, y in Q, one of the following conditions is satisfied.

(1) $\text{atoms}(x) \subseteq \text{atoms}(y)$

(2) $\text{atoms}(x) \supseteq \text{atoms}(y)$

(3) $\text{atoms}(x) \cap \text{atoms}(y) = \emptyset$

A conjunctive query Q is *q-hierarchical* if it is hierarchical and for all variables $x, y \in \text{vars}(Q)$ the following is satisfied: if $\text{atoms}(x) \subsetneq \text{atoms}(y)$ holds and x is in the head of Q, then y is also in the head of Q.

For brevity, we denote by CQ, ACQ, CCQ, HCQ, and QHCQ the classes of conjunctive queries in general and those conjunctive queries that are acyclic, free-connex acyclic, hierarchical or q-hierarchical, respectively. We note that each q-hierarchical query is free-connex acyclic [20, Proposition 4.25] and each hierarchical query is acyclic.[1]

Views and rewritings. A view V over a schema $\mathcal{S}$ is just a query over schema $\mathcal{S}$.[2] A set $\mathcal{V} = \{V_1, \ldots, V_k\}$ of views induces, for each database D over schema $\mathcal{S}$, the $\mathcal{V}$-defined *database* $\mathcal{V}(D) = V_1(D) \cup \cdots \cup V_k(D)$.

In this paper, we consider only views that are conjunctive queries. We furthermore assume that every view defines its own relation in the derived database. We denote the set of relation names of the heads of the views in $\mathcal{V}$ by $\mathcal{S}_\mathcal{V}$.

In a nutshell, a rewriting of a query Q over a set $\mathcal{V}$ of views is a query over $\mathcal{S}_\mathcal{V}$ that is meant to yield, for each database D, the same result over $\mathcal{V}(D)$ as Q does over D.

---

[1] The latter inclusion is mentioned in, e.g. [19, 21]; it also follows readily from the former inclusion. For acyclicity the head of a query is of no concern, and the Boolean variant of a hierarchical query is always q-hierarchical by definition, and, thus, free-connex acyclic which implies the existence of a join tree for the body of the original query.

[2] They are called views due to their special role which distinguishes them from "normal" queries.

▶ **Definition 2.2.** Let Q be a query, $\mathcal{V}$ a set of views, and Q' a query over $\mathcal{S}_\mathcal{V}$. We call Q' a *$\mathcal{V}$-rewriting of Q* if, for every database D, it holds that $Q'(\mathcal{V}(D)) = Q(D)$.

3 The Rewriting Problem

In this section, we define the rewriting problem, recall some known results about its complexity and discuss some relevant concepts that were used to tackle it.

▶ **Definition 3.1.** Let $\mathbb{V}$, $\mathbb{Q}$, and $\mathbb{R}$ be classes of conjunctive queries. The *rewriting problem* for $\mathbb{V}$, $\mathbb{Q}$, and $\mathbb{R}$, denoted $\textsc{Rewr}(\mathbb{V}, \mathbb{Q}, \mathbb{R})$ asks, upon input of a query $Q \in \mathbb{Q}$ and a (finite) set $\mathcal{V} \subseteq \mathbb{V}$ of views, whether there is a $\mathcal{V}$-rewriting of Q in the class $\mathbb{R}$. We write $\textsc{Rewr}^k(\mathbb{V}, \mathbb{Q}, \mathbb{R})$ for the restriction, where the arity of the database schema is bounded by k.

In general, the rewriting problem for conjunctive queries is NP-complete.

▶ **Theorem 3.2** ([25, Theorem 3.10])**.** $\textsc{Rewr}(\mathsf{CQ}, \mathsf{CQ}, \mathsf{CQ})$ *is* NP*-complete.*

There is a straightforward, albeit in general inefficient, way of finding a rewriting for a conjunctive query and views. In fact, it can be shown that, for given Q and $\mathcal{V}$, a rewriting Q' over $\mathcal{V}$ exists, if and only if a certain canonical query $\mathsf{canon}(Q, \mathcal{V})$ exists and is a rewriting.

This *canonical candidate* $\mathsf{canon}(Q, \mathcal{V})$ can be obtained by evaluating $\mathcal{V}$ on the canonical database $\mathsf{canon}(Q)$, which in turn is defined as the set of body atoms of Q viewed as facts, in which the variables are considered as fresh constants. The canonical candidate has the same head as Q and its body is just $\mathcal{V}(\mathsf{canon}(Q))$.[3] If the canonical candidate turns out to be a rewriting, then we often call it *canonical rewriting*.

An important detail should be mentioned here: the canonical candidate does not always exist, e.g., if $\mathcal{V}(\mathsf{canon}(Q))$ does not contain all head variables of Q or it is outright empty. In that case, there is no rewriting.

▶ **Proposition 3.3** ([26, Proposition 5.1])**.** *Let Q be a CQ and $\mathcal{V}$ a set of views. If there is a $\mathcal{V}$-rewriting of Q, then the canonical candidate $\mathsf{canon}(Q, \mathcal{V})$ is such a rewriting.*

▶ **Example 3.4.** Let us consider the query Q defined by

$$H(x, y, z) \leftarrow C(x, y, z),\ R(x, y),\ S(y, z),\ T(z, x)$$

and the views

$$V_1(u, v, w) \leftarrow C(u, v, w) \quad \text{and} \quad V_2(x, y, z, u) \leftarrow R(x, y),\ S(y, z),\ T(z, u).$$

Evaluating the views V_1 and V_2 on the canonical database $\mathsf{canon}(Q)$ of Q yields the result $\{V_1(x, y, z), V_2(x, y, z, x)\}$. Thus, the query Q' defined by

$$H(x, y, z) \leftarrow V_1(x, y, z), V_2(x, y, z, x)$$

is the canonical candidate, which happens to be an actual $\mathcal{V}$-rewriting.

We will need the following notions throughout the paper.

▶ **Definition 3.5** (View Application)**.** An *application* of a view V is a substitution $\alpha\colon \mathrm{vars}(V) \to \mathsf{var}$ that does not unify any quantified variable x of V with another variable of V.

[3] But here the variables are again considered as variables.

We say that a collection $\alpha_1, \ldots, \alpha_m$ of applications for views $V_1, \ldots, V_m$ fulfils *quantified variable disjointness*, if for all $i, j \in \{1, \ldots, m\}$, each quantified variable x of V_i and each variable y of V_j with $i \neq j$, it holds $\alpha_i(x) \neq \alpha_j(y)$. This ensures that, for all $i, j \in \{1, \ldots, m\}$ with $i \neq j$, the bodies $\alpha_i(\text{body}(V_i))$ and $\alpha_j(\text{body}(V_j))$ do not share any variables that do not occur in $\text{vars}(\alpha_i(\text{head}(V_i))) \cap \text{vars}(\alpha_j(\text{head}(V_j)))$.

▶ **Definition 3.6** (Expansion)**.** Let $\mathcal{V}$ be a set of views over a schema $\mathcal{S}$ and let Q' be a query with $\text{body}(Q') = \{A'_1, \ldots, A'_m\}$ over the schema $\mathcal{S}_\mathcal{V}$. Furthermore, let, for each $i \in \{1, \ldots, m\}$, α_i be an application and V_i be a view such that $A'_i = \alpha_i(\text{head}(V_i))$, such that $\alpha_1, \ldots, \alpha_m$ fulfil quantified variable disjointness.

The *expansion of Q' w.r.t. $\mathcal{V}$ and $\alpha_1, \ldots, \alpha_m$* is the query that has the same head as Q' and body $\bigcup_{i=1}^{m} \alpha_i(\text{body}(V_i))$.

Since the applications $\alpha_1, \ldots, \alpha_m$ in Definition 3.6 are uniquely determined up to renaming of quantified variables, we usually do not mention them explicitly and just speak of an expansion of Q' w.r.t. $\mathcal{V}$.

We note that the expansion of Q' can be directly compared with Q since it is over the same schema. In fact, Q' is a $\mathcal{V}$-rewriting of Q if and only if the expansion is equivalent to Q. In Example 3.4, this is obviously the case, since (the only) expansion of Q' w.r.t. $\mathcal{V}$ even coincides with Q.

Even though the canonical candidate can be of exponential size in $|Q| + |\mathcal{V}|$, there is always a rewriting of polynomial size if a rewriting exists. In fact, at most n view applications are needed if the body of query Q contains n atoms [25, Lemma 3.5]. Therefore, and because equivalence of CQ can be tested in NP, $\textsc{Rewr}(\mathsf{CQ}, \mathsf{CQ}, \mathsf{CQ})$ is in NP.

However, for acyclic views of bounded arity and acyclic queries, the situation is much better (cf. the full version [15] for a short proof). For a class $\mathbb{V}$ of views, we write $\mathbb{V}^k$ for the restriction to views of arity at most k.

▶ **Proposition 3.7.** *For every $k \geq 0$, $\textsc{Rewr}(\mathsf{ACQ}^k, \mathsf{ACQ}, \mathsf{CQ})$ is in polynomial time.*

Chekuri and Rajaraman have shown that the rewriting problem is in P for acyclic queries without self-joins and arbitrary views using a similar idea [10, Theorem 5].

However, even for Boolean views and databases over a small fixed schema, it does not suffice to restrict only the query or only the views to acyclic queries.

▶ **Proposition 3.8.** $\textsc{Rewr}^k(\mathsf{CQ}^0, \mathsf{ACQ}, \mathsf{CQ})$ *and* $\textsc{Rewr}^k(\mathsf{ACQ}^0, \mathsf{CQ}, \mathsf{CQ})$ *are* NP*-hard, for every $k \geq 3$. This even holds for Boolean queries, view sets with only one Boolean view and a schema with two relations of maximum arity 3.*

The proof is given in the full version [15].

4 A Characterisation

In this section we give a characterisation of rewritability of a query Q with respect to a set $\mathcal{V}$ of views. It is in terms of a partition of the atoms of $\text{body}(Q)$, where each set of the partition can be matched with a view in a specific way. We refer to such partitions as *cover partitions* and the matches as *cover descriptions*. The characterisation is very similar to other such characterisations in the literature [17, 2], in particular to "MiniCon Descriptions" [28]. However, in its specific form and notation it is tailored for our needs in the subsequent sections. We will discuss the relationship of our characterisation with others further below.

Next we define the notions of *cover descriptions* and *cover partitions*. Let Q be a conjunctive query. For a set $\mathcal{A} \subseteq \text{body}(Q)$ we define $\text{bvars}_Q(\mathcal{A})$ as the set of *bridge variables* of $\mathcal{A}$, that occur in $\mathcal{A}$ as well as in the head of Q or in some atom of Q not in $\mathcal{A}$. More formally, $\text{bvars}_Q(\mathcal{A}) = \text{vars}(\mathcal{A}) \cap (\text{vars}(\text{head}(Q)) \cup \text{vars}(\text{body}(Q) \setminus \mathcal{A}))$.

▶ **Definition 4.1** (Cover Description). A *cover description* C for a query Q is a tuple $(\mathcal{A}, V, \alpha, \psi)$ where
- $\mathcal{A}$ is a subset of body(Q),
- V is a view,
- α is an application of V, and
- ψ is a mapping from vars$(\alpha(V))$ to vars(Q),

such that
(1) $\mathcal{A} \subseteq \alpha(\text{body}(V))$,
(2) $\text{bvars}_Q(\mathcal{A}) \subseteq \alpha(\text{vars}(\text{head}(V)))$,
(3) ψ is a body homomorphism from $\alpha(V)$ to Q, and
(4) ψ is the identity on vars$(\mathcal{A})$.

▶ **Example 4.2.** Consider the query Q given by the rule

$$H(x, y, z) \leftarrow R(x, y, z), T(x, v), F(v), E(w), S(w, z)$$

and the following views.

$$V_1(x_1, y_1, w_1) \leftarrow R(x_1, y_1, u_1), T(x_1, v_1), F(v_1), E(w_1), S(w_1, u_1)$$
$$V_2(x_2, y_2, z_2) \leftarrow R(x_2, y_2, z_2), F(v_2)$$
$$V_3(w_3, z_3) \leftarrow S(w_3, z_3), E(w_3)$$

The tuple $C_1 = (\mathcal{A}_1, V_1, \alpha_1, \psi_1)$ with $\mathcal{A}_1 = \{T(x, v), F(v)\}$,

$$\alpha_1 = \{x_1 \mapsto x, y_1 \mapsto y', u_1 \mapsto u', v_1 \mapsto v, w_1 \mapsto w'\}, \text{ and}$$
$$\psi_1 = \{x \mapsto x, y' \mapsto y, u' \mapsto z, v \mapsto v, w' \mapsto w\}$$

is a cover description for Q with $\text{bvars}_Q(\mathcal{A}_1) = \{x\}$. Although ψ_1 could be chosen as id (by adapting α_1 accordingly), we will see in Example 4.4, that this is not always desirable.

Now we can simply characterise rewritability of a query Q by the existence of a partition of body(Q) whose subsets have cover descriptions with views from $\mathcal{V}$.

▶ **Definition 4.3.** Let Q be a query and $\mathcal{V}$ be a set of views. A collection $\mathcal{C} = C_1, \ldots, C_m$ of cover descriptions $C_i = (\mathcal{A}_i, V_i, \alpha_i, \psi_i)$ for Q with $V_i \in \mathcal{V}$ is a *cover partition* for Q over $\mathcal{V}$ if the sets $\mathcal{A}_1, \ldots, \mathcal{A}_m$ constitute a partition of body(Q).

We call a cover partition *consistent* if variables of any $\alpha_j(V_j)$ are in the range of any other α_i, only if they also appear in $\text{bvars}_Q(\mathcal{A}_j)$. We note that, since each α_i is a view application, in a consistent cover partition, the applications obey quantified variable disjointness.[4]

▶ **Example 4.4** (Continuation of Example 4.2). Let $C_1 = (\mathcal{A}_1, V_1, \alpha_1, \psi_1)$ be the cover description defined in Example 4.2. In addition, we consider the cover descriptions C_2 and C_3 with $C_i = (\mathcal{A}_i, V_i, \alpha_i, \psi_i)$ for $i \in \{2, 3\}$ where

$$\mathcal{A}_2 = \{R(x, y, z)\}, \qquad \mathcal{A}_3 = \{E(w), S(w, z)\},$$
$$\alpha_2 = \{x_2 \mapsto x, y_2 \mapsto y, z_2 \mapsto z, v_2 \mapsto v\}, \qquad \alpha_3 = \{w_3 \mapsto w, z_3 \mapsto z\},$$

[4] For the sake of contradiction, assume two quantified variables x and y are mapped to the same variable z by two different view applications α_i and α_j, respectively. Then, due to consistency, z is in $\text{bvars}(\mathcal{A}_i)$ and, thus, in $\alpha_i(\text{head}(V_i))$ due to Definition 4.1(2). It follows that α_i unifies x with a head variable. But this is a contradiction to α_i being a view application.

and $\psi_2 = \psi_3 = \mathrm{id}$. The cover descriptions C_1, C_2, C_3 constitute a cover partition for Q over $\{V_1, V_2, V_3\}$. It is, however, not consistent, since v is in the range of α_1 and α_2, but not in $\mathrm{bvars}(\mathcal{A}_1)$. Replacing α_2 and ψ_2 by mappings $\hat{\alpha}_2$ and $\hat{\psi}_2$ with $\hat{\alpha}_2(v_2) = \hat{v}$ and $\hat{\psi}_2(\hat{v}) = v$ that agree with α_2 and ψ_2 on all other variables, respectively, yields a consistent cover partition. Note that there is no consistent cover partition of the form $(\mathcal{A}_1, V_1, \alpha_1', \mathrm{id})$ because necessarily $\alpha_1'(w_1) = w$ would hold and thus w would be in the range of α_1' and α_3.

A consistent cover partition $\mathcal{C}$ induces a query $Q_\mathcal{C}$ and an expansion $Q'_\mathcal{C}$ as follows. We note first that each variable in $\mathrm{head}(Q)$ occurs in at least one of the sets $\mathcal{A}_i$ and thus in some set $\mathrm{bvars}_Q(\mathcal{A}_i)$. Therefore, Condition (2) guarantees that each head variable of Q occurs in some set $\alpha_i(\mathrm{head}(V_i))$ and thus a cover partition $\mathcal{C} = C_1, \ldots, C_m$ with $C_i = (\mathcal{A}_i, V_i, \alpha_i, \psi_i)$ induces a query $Q_\mathcal{C}$ with

$$\mathrm{head}(Q_\mathcal{C}) = \mathrm{head}(Q) \text{ and } \mathrm{body}(Q_\mathcal{C}) = \{\alpha_i(\mathrm{head}(V_i)) \mid 1 \leq i \leq m\}.$$

Likewise, and thanks to quantified variable disjointness, it induces an expansion $Q'_\mathcal{C}$ with

$$\mathrm{head}(Q'_\mathcal{C}) = \mathrm{head}(Q) \text{ and } \mathrm{body}(Q'_\mathcal{C}) = \{\alpha_i(\mathrm{body}(V_i)) \mid 1 \leq i \leq m\}.$$

Next, we show that the existence of a cover partition indeed characterises rewritability.

▶ **Theorem 4.5.** *Let Q be a minimal query and $\mathcal{V}$ be a set of views. The following three statements are equivalent.*
(a) *Q is $\mathcal{V}$-rewritable.*
(b) *There is a cover partition $\mathcal{C}$ for Q over $\mathcal{V}$.*
(c) *There is a consistent cover partition $\mathcal{C}$ for Q over $\mathcal{V}$.*
If $\mathcal{C}$ is a consistent cover partition, then Q_C is a $\mathcal{V}$-rewriting of Q.

Proof ideas. To show that (c) implies (a) we consider a consistent cover partition $\mathcal{C} = C_1, \ldots, C_m$ for Q over $\mathcal{V}$, where, for each i, $C_i = (\mathcal{A}_i, V_i, \alpha_i, \psi_i)$. We prove that the query $Q_\mathcal{C}$ induced by $\mathcal{C}$ is a $\mathcal{V}$-rewriting of Q: first, since the sets $\mathcal{A}_1, \ldots, \mathcal{A}_m$ partition $\mathrm{body}(Q)$ and thanks to Condition (1), the identity mapping is a homomorphism from Q into Q'_C. Therefore, it suffices to show that the mapping h' from $Q'_\mathcal{C}$ into Q, defined as the union of the mappings ψ_i, is a homomorphism.

To show that (a) implies (b) we assume that Q has a $\mathcal{V}$-rewriting Q_R with an expansion Q_E. We show that we can assume that the body of Q is a subset of Q_R and that there is a reverse homomorphism h' which is the identity on all variables of Q. That is, we have $\mathrm{head}(Q) = \mathrm{head}(Q_E) = \mathrm{head}(Q_R)$ as well as $\mathrm{body}(Q) \subseteq \mathrm{body}(Q_E)$.

Let $\alpha_1, \ldots, \alpha_m$ be applications of views $V_1, \ldots, V_m$ that result in expansion Q_E, that is, $\mathrm{body}(Q_E) = \bigcup_{i=1}^{m} \mathrm{body}(\alpha_i(V_i))$, where the applications fulfil the quantified variable disjointness property. These applications induce a partition $\mathcal{A}_1, \ldots, \mathcal{A}_m$ of $\mathrm{body}(Q)$ via $\mathcal{A}_i = \{A \in \mathrm{body}(Q) \mid i \text{ is minimal such that } A \in \alpha_i(\mathrm{body}(V_i))\}$. We show then that this partition induces a cover partition.

Finally, we show that (b) implies (c) by renaming some variables. We note that this step does not change the underlying partition of $\mathrm{body}(Q)$. ◀

The full proof is given in the full version [15].

▶ Remark 4.6. We note that the requirement in Theorem 4.5 for Q to be minimal is only seemingly a restriction in our setting, since we apply it only to acyclic queries. If Q is acyclic but not minimal, an equivalent minimal query Q' can be computed in polynomial time by

iteratively removing atoms from its body [9, 10]. Moreover, it is guaranteed that the minimal query Q' is also acyclic (we believe this to be folklore, it follows readily from more general results, cf. for instance [5]).

The same is true for free-connex acyclic queries: every homomorphism from Q to Q' is also a homomorphism from $\text{body}(Q) \cup \{\text{head}(Q)\}$ to $\text{body}(Q') \cup \{\text{head}(Q')\}$ and vice versa, since the relation symbol of $\text{head}(Q) = \text{head}(Q')$ does not occur in $\text{body}(Q')$. Thus, Q' is minimal if and only if the Boolean query whose body is $\text{body}(Q') \cup \{\text{head}(Q')\}$ is minimal. Therefore, if $\text{body}(Q) \cup \{\text{head}(Q)\}$ is acyclic, so is $\text{body}(Q') \cup \{\text{head}(Q')\}$. In other words, if Q is free-connex acyclic, then Q' is free-connex acyclic as well.

For hierarchical and q-hierarchical queries the same holds: it is easy to see that removing atoms does not change the conditions in their respective definitions.

5 Towards Acyclic Rewritings

In this section, we turn our focus to the main topic of this paper: acyclic rewritings and the decision problem that asks whether such a rewriting exists. We study the complexity of $\textsc{Rewr}(\mathbb{V}, \mathbb{Q}, \mathsf{ACQ})$ for the case that $\mathbb{V}$ and $\mathbb{Q}$ are the class of conjunctive queries and for various subclasses. It will be helpful to analyse the case that $\mathbb{Q} = \mathsf{ACQ}$ and $\mathbb{V} = \mathsf{CQ}$ first.

5.1 A Characterisation of Acyclic Rewritability for Acyclic Queries

The following example illustrates that, even if an acyclic rewriting exists, the canonical rewriting need not be acyclic. Furthermore, it may be that each "sub-rewriting" of the canonical candidate is cyclic or not a rewriting, and thus none of them is an acyclic rewriting.

▶ **Example 5.1.** Consider the query Q given by the rule

$$H() \leftarrow R_1(x), R_2(y), S(x, z), T_1(z), T_2(y)$$

and the following views $\mathcal{V} = \{V_1, V_2, V_3\}$.

$$V_1(u, v) \leftarrow R_1(u), R_2(v) \qquad V_2(u, v) \leftarrow S(u, v) \qquad V_3(u, v) \leftarrow T_1(u), T_2(v)$$

The canonical candidate $Q_R = H() \leftarrow V_1(x, y), V_2(x, z), V_3(z, y)$ is a $\mathcal{V}$-rewriting, but it is *cyclic*. Each query whose body is a proper subset of the body of Q_R is not a $\mathcal{V}$-rewriting for Q. However, an *acyclic* $\mathcal{V}$-rewriting for Q exists. One such rewriting is the query Q'_R.

$$H() \leftarrow V_1(x, y), V_2(x, z), V_3(z, y'), V_3(z', y)$$

Even though the example suggests that general and acyclic rewritings can be seemingly unrelated, it turns out that there is a close connection between them. In fact, we will show next that an acyclic $\mathcal{V}$-rewriting exists if and only if an arbitrary $\mathcal{V}$-rewriting exists. Furthermore, from an arbitrary rewriting Q_R, an acyclic one Q_C can be constructed.

Towards a proof, we study the relationship between the target query Q and the view applications that can occur in any rewriting of Q more closely, to determine the circumstances under which a decomposition of view atoms is possible.

▶ **Example 5.2** (Continuation of Example 5.1)**.** In the SPJ-algebra[5], the V_3-atom in Q_R can be expressed as the SPJ-expression $\pi_{z,y}(V_3(z, y))$, whereas the two V_3-atoms in Q'_R can be expressed as $\pi_{z,y}(V_3(z, y') \bowtie V_3(z', y))$. When expanded by the SPJ-expressions that define

[5] SPJ-algebra refers to the select-project-join-algebra.

V_3, these two expressions are equivalent. That is, the acyclic rewriting can be obtained from the canonical rewriting by replacing an atom by a set of atoms that is equivalent with respect to view expansions.[6]

In Example 5.1 the acyclic rewriting was obtained from the canonical rewriting by replacing a view atom by a set that contained one view atom for each connected component of V_3. The following example shows that the required modifications can be more involved.

▶ **Example 5.3.** Consider the query Q given by the rule

$$H(x, y, z) \leftarrow R(x, y, z), E_1(x), E_2(y), E_3(w), S(w, z)$$

and the following views.

$$V_1(x, y, w) \leftarrow R(x, y, v), E_1(x), E_3(w), S(w, v) \qquad V_2(x, y, z) \leftarrow R(x, y, z), E_2(y)$$
$$V_3(w, z) \leftarrow S(w, z), E_3(w)$$

The canonical candidate $H(x, y, z) \leftarrow V_1(x, y, w), V_2(x, y, z), V_3(w, z)$ is a cyclic $\{V_1, V_2, V_3\}$-rewriting of Q. In contrast to Example 5.1, all view bodies are connected. Therefore, replacing a view atom by a set of representatives of connected components of some views does not yield an acyclic rewriting. However, there is an acyclic $\{V_1, V_2, V_3\}$-rewriting of Q, namely $H(x, y, z) \leftarrow V_1(x, y', w'), V_2(x, y, z), V_3(w, z)$.

Now we turn to the main result of this section. Its proof is given in the full version [15].

▶ **Theorem 5.4.** *Let Q be a (free-connex) acyclic conjunctive query and $\mathcal{V}$ a set of CQ-views. If Q is $\mathcal{V}$-rewritable, then there is an (free-connex) acyclic $\mathcal{V}$-rewriting of Q.*

The proof shows that a given cover partition for Q can be modified such that each of its cover descriptions induces a connected sub-tree of a join tree of Q. Then it is shown that by collapsing these sub-trees into nodes, a join tree for the rewriting induced by the modified cover partition is obtained.

Theorem 5.4 delivers good news as well as bad news. The good news is that, since the proofs of Theorem 5.4 and Theorem 4.5 are constructive, we altogether have a procedure to construct an acyclic rewriting from an arbitrary rewriting Q_R if the query Q is acyclic. Furthermore, if a homomorphism from an expansion of Q_R to Q can be computed in polynomial time, the overall procedure can be performed in polynomial time. In particular, we get the following collary.

▶ **Corollary 5.5.** *For every k, $\textsc{Rewr}(\mathsf{ACQ}^k, \mathsf{ACQ}, \mathsf{ACQ})$ is in polynomial time and an acyclic rewriting can be computed in polynomial time (if it exists).*

That the decision problem is in polynomial time follows immediately from Proposition 3.7 and Theorem 5.4. Moreover, the canonical candidate for an acyclic query Q and a set $\mathcal{V}$ of acyclic views with bounded arity can be computed in polynomial time, since the size of $\mathcal{V}(\mathsf{canon}(Q))$ is bounded by a polynomial and query evaluation for acyclic queries is in polynomial time [30, 1]. In case the canonical candidate is a rewriting, valuations witnessing $\mathcal{V}(\mathsf{canon}(Q))$ can be computed in polynomial time [23] and combined into a homomorphism from an expansion of the rewriting to Q. Thus, an acyclic rewriting can then be efficiently constructed by the above procedure.

[6] We note that one could also replace the V_1-atom in Q_R.

However, this leaves open the complexity of $\textsc{Rewr}(\mathsf{ACQ}, \mathsf{ACQ}, \mathsf{ACQ})$ (as well as that of $\textsc{Rewr}(\mathsf{ACQ}, \mathsf{ACQ}, \mathsf{CQ})$), and of their restrictions to schemas of bounded arity. The bad news is that, since $\textsc{Rewr}(\mathbb{V}, \mathsf{ACQ}, \mathsf{CQ})$ and $\textsc{Rewr}(\mathbb{V}, \mathsf{ACQ}, \mathsf{ACQ})$ are basically the same problem, lower bounds on $\textsc{Rewr}(\mathbb{V}, \mathsf{ACQ}, \mathsf{CQ})$ transfer to $\textsc{Rewr}(\mathbb{V}, \mathsf{ACQ}, \mathsf{ACQ})$. In fact, we get the following, due to NP-hardness of $\textsc{Rewr}^k(\mathsf{CQ}, \mathsf{ACQ}, \mathsf{CQ})$ in the general case (Proposition 3.8).[7]

▶ **Corollary 5.6.** *For every $k \geq 3$, the problem $\textsc{Rewr}^k(\mathsf{CQ}, \mathsf{ACQ}, \mathsf{ACQ})$ and, therefore, also $\textsc{Rewr}^k(\mathsf{CQ}, \mathsf{CQ}, \mathsf{ACQ})$ is* NP*-hard.*

In the next part of this section, we resolve the complexity of $\textsc{Rewr}(\mathsf{ACQ}, \mathsf{ACQ}, \mathsf{CQ})$ and $\textsc{Rewr}(\mathsf{ACQ}, \mathsf{ACQ}, \mathsf{ACQ})$ and their restrictions to schemas of bounded arity.

5.2 The Complexity of Acyclic Rewritability for Acyclic Queries

It may be tempting to assume that, since acyclic queries are so well-behaved in general, it should be tractable to decide whether for an acyclic query and a set of acyclic views there exists an acyclic rewriting. However, as we show next, this is (probably) not the case, and this surprising finding even holds for the even better behaved hierarchical queries as well.

▶ **Theorem 5.7.** *The problems $\textsc{Rewr}^k(\mathsf{ACQ}, \mathsf{ACQ}, \mathsf{CQ})$ and $\textsc{Rewr}^k(\mathsf{ACQ}, \mathsf{ACQ}, \mathsf{ACQ})$, as well as $\textsc{Rewr}^k(\mathsf{HCQ}, \mathsf{HCQ}, \mathsf{CQ})$ are* NP*-complete, for $k \geq 3$.[8] The lower bounds even hold for instances with a single view.*

We will see in Section 7 that Theorem 5.4 has an analogue for hierarchical queries from which it can be concluded that deciding the existence of a hierarchical rewriting, given a hierarchical query and hierarchical views, is still NP-complete.

Theorem 5.7 will easily follow from NP-hardness of a seemingly simpler problem. From the characterisation in Theorem 4.5, we already know that deciding the existence of a rewriting is the same as deciding the existence of a cover partition. We show next that it is even NP-hard to decide the existence of a *cover description*, given a query, a set of atoms, and a single view.

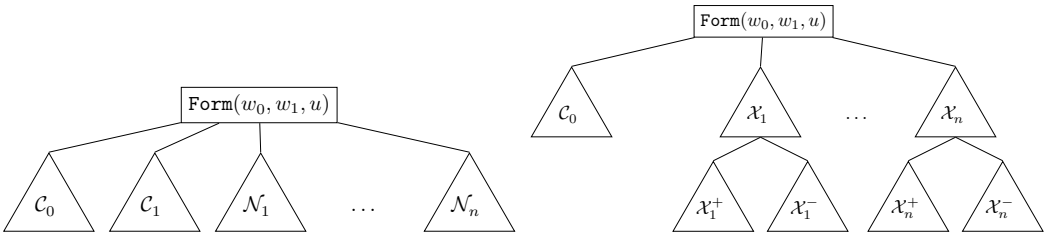
▶ **Definition 5.8.** Let $\mathbb{V} \subseteq \mathsf{CQ}$ and $\mathbb{Q} \subseteq \mathsf{CQ}$ be classes of conjunctive queries. The *cover description problem* for $\mathbb{V}$ and $\mathbb{Q}$, denoted $\textsc{CovDesc}(\mathbb{V}, \mathbb{Q})$ asks, upon input of a query $Q \in \mathbb{Q}$, a subset $\mathcal{A} \subseteq \text{body}(Q)$ and a view $V \in \mathbb{V}$, whether mappings α and ψ exist such that $(\mathcal{A}, V, \alpha, \psi)$ is a cover description.

▶ **Theorem 5.9.** $\textsc{CovDesc}(\mathsf{ACQ}, \mathsf{ACQ})$ *is* NP*-hard. Indeed, even $\textsc{CovDesc}(\mathsf{HCQ}, \mathsf{HCQ})$ is* NP*-hard and even if the input is restricted to $\mathcal{A} = \text{body}(Q)$.*

Proof. We reduce problem 3SAT to $\textsc{CovDesc}(\mathsf{HCQ}, \mathsf{HCQ})$. This lower bound directly translates to $\textsc{CovDesc}(\mathsf{ACQ}, \mathsf{ACQ})$ since every hierarchical query is acyclic. For a formula f in 3CNF, we describe how a query Q and a view V can be derived in polynomial time such that both Q and V are hierarchical and such that f is satisfiable if and only if there are mappings α and ψ such that $(\text{body}(Q), V, \alpha, \psi)$ is a cover description for Q.

[7] We note that Corollary 5.6 also follows from the proof of Proposition 3.8, since the canonical candidate constructed in that proof is trivially acyclic. Indeed, NP-hardness of $\textsc{Rewr}(\mathsf{ACQ}, \mathsf{CQ}, \mathsf{ACQ})$ is also implied.

[8] These results hold for schemas of unbounded arity, as well.



(a) Structure of query $\boldsymbol{Q}$, induced by formula $\boldsymbol{f}$. **(b)** Structure of view $\boldsymbol{V}$, induced by formula $\boldsymbol{f}$.

Figure 1 Structure of the query $\boldsymbol{Q}$ and the view $\boldsymbol{V}$, induced by formula $\boldsymbol{f}$.

Construction. Let $f = f_1 \wedge \cdots \wedge f_k$ be a propositional formula in 3CNF over variables $x_1, \ldots, x_n$ in clauses $f_1, \ldots, f_k$, where $f_j = (\ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3})$ for each $j \in \{1, \ldots, k\}$.

We start with the construction of query Q. This query is Boolean, with head $H()$, and uses only three variables w_0, w_1 and u, where the first two are intended to represent the truth values false and true. The structure of Q is depicted in Figure 1a. The body of Q is defined as the union

$$\{\texttt{Form}(w_0, w_1, u)\} \uplus \mathcal{C}_0 \uplus \mathcal{C}_1 \uplus \mathcal{N}_1 \uplus \cdots \uplus \mathcal{N}_n$$

of the following sets of atoms.

- Set $\mathcal{C}_0$ contains an atom $\texttt{C}_j(w_0, w_1, u)$ for each clause f_j in f.

  Intuitively, these atoms represent *unsatisfied* clauses as opposed to the next atoms that represent *satisfied* clauses. Note that these atoms differ in the order of variables w_0 and w_1 only.
- Set $\mathcal{C}_1$ contains an atom $\texttt{C}_j(w_1, w_0, u)$ for each clause f_j in f.
- There is a set $\mathcal{N}_i$ for each variable x_i in formula f. The sets differ only in the name of the relation they address and contain two atoms $\texttt{Neg}_i(w_0, w_1, u)$ and $\texttt{Neg}_i(w_1, w_0, u)$ each.

Query Q is hierarchical since every atom in each of the sets above refers to all three variables w_0, w_1 and u. Thus, query Q is acyclic in particular.

We now proceed with the construction of view V, which refers to the same relations as query Q. Like query Q, the view uses variables w_0, w_1 and u but also additional variables for the propositions $x_1, \ldots, x_n$ in formula f. For each proposition x_i, there are two variables x_i and $\bar{x}_i$, intended to represent the positive and negated literal over the proposition, respectively. With the exception of variable u, all other variables are in the head $H(w_0, w_1, x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n)$ of view V. The body of V, whose structure is depicted in Figure 1b, is defined as the union

$$\{\texttt{Form}(w_0, w_1, u)\} \uplus \mathcal{C}_0 \uplus \left(\mathcal{X}_1 \uplus \mathcal{X}_1^+ \uplus \mathcal{X}_1^-\right) \uplus \cdots \uplus \left(\mathcal{X}_n \uplus \mathcal{X}_n^+ \uplus \mathcal{X}_n^-\right)$$

of sets of atoms, where the first two sets $\{\texttt{Form}(w_0, w_1, u)\}$ and $\mathcal{C}_0$ are the same as above. The bodies of query Q and view V thus differ in the sets $\mathcal{C}_1$ and $\mathcal{N}_1, \ldots, \mathcal{N}_n$ of atoms on the one hand as well as $\mathcal{X}_1, \ldots, \mathcal{X}_n$ and $\mathcal{X}_1^+, \ldots, \mathcal{X}_n^+$ and $\mathcal{X}_1^-, \ldots, \mathcal{X}_n^-$ on the other hand. Sets $\mathcal{N}_1, \ldots, \mathcal{N}_n$ and $\mathcal{X}_1, \ldots, \mathcal{X}_n$ are intended to guide the assignment of propositions while sets $\mathcal{X}_1^+, \ldots, \mathcal{X}_n^+$ and $\mathcal{X}_1^-, \ldots, \mathcal{X}_n^-$ represent the occurrences of literals in the clauses of formula f and set $\mathcal{C}_1$ represents the aim to satisfy all of them. The new sets are defined as follows.

- For each proposition x_i, set $\mathcal{X}_i$ contains atoms $\texttt{Neg}_i(x_i, \bar{x}_i, u)$ and $\texttt{Neg}_i(\bar{x}_i, x_i, u)$, intended to enforce the mapping of $(x_i, \bar{x}_i)$ to either (w_0, w_1) or (w_1, w_0), that is, to "complementary truth values".

- For each positive literal x_i and each clause f_j that contains literal x_i, set $\mathcal{X}_i^+$ contains an atom $\mathtt{C}_j(x_i, \bar{x}_i, u)$.
- For each negated literal $\bar{x}_i$ and each clause f_j that contains literal $\neg x_i$, set $\mathcal{X}_i^-$ contains an atom $\mathtt{C}_j(\bar{x}_i, x_i, u)$.

Thus defined, the view is also hierarchical as shown in the following. Every atom in every set contains variable u. Therefore, it follows that $\text{atoms}(x) \subseteq \text{atoms}(u)$ for all $x \in \text{vars}(V)$. Furthermore, all atoms in $\{\mathtt{Form}(w_0, w_1, u)\}$ and $\mathcal{C}_0$ additionally contain both variables w_0 and w_1 and they are the only atoms to contain these variables. Thus, we have that $\text{atoms}(w_0) = \text{atoms}(w_1)$ and $\text{atoms}(y) \cap \text{atoms}(w_0) = \emptyset$ and $\text{atoms}(y) \cap \text{atoms}(w_1) = \emptyset$ for all $y \in \text{vars}(V) \setminus \{u, w_0, w_1\}$. Similarly, for each $i \in \{1, \ldots, n\}$, all atoms in $\mathcal{X}_i \cup \mathcal{X}_i^+ \cup \mathcal{X}_i^-$ contain both variables x_i and $\bar{x}_i$, which do not occur in any other atom. It follows that $\text{atoms}(x_i) = \text{atoms}(\bar{x}_i)$ and $\text{atoms}(\bar{x}_i) \cap \text{atoms}(y) = \emptyset$ and $\text{atoms}(x_i) \cap \text{atoms}(y) = \emptyset$ for all $y \in \text{vars}(V) \setminus \{x_i, \bar{x}_i, u\}$ and $i \in \{1, \ldots, n\}$. Therefore, we obtain that each pair of variables in V satisfies at least one of the conditions in Definition 2.1 and thus V is hierarchical – and acyclic in particular.

For the correctness argument, however, the structure of the join tree of view V, depicted in Figure 1b is probably more helpful. We close the description of the construction by remarking that query and view can be computed in polynomial time for any given propositional formula in 3CNF. The correctness of this construction is shown in the full version [15]. ◀

The proof of Theorem 5.7 can now be stated easily.

Proof of Theorem 5.7. The upper bound follows from Theorem 3.2. For the lower bound, we observe that the proof of Theorem 5.9 shows that query Q has a V-rewriting *if* the formula f is satisfiable. Moreover, the query cannot be covered (non-redundantly) by multiple applications of this view because variable u is no head variable of view V. Thus, we can conclude that Q has a V-rewriting *only if* formula f is satisfiable. Hence, NP-hardness transfers to the rewriting problem. ◀

Notably, the complexity does not result from the restriction to rewritings that are acyclic since the last argument does not depend on the acyclicity.

The fact that, for Theorem 5.7, a single view application is sufficient if there is any rewriting, allows to draw yet another conclusion for a related problem, defined next. In the following, a *selection* is a query that can be expressed by a composition of select-operators.

▶ **Definition 5.10.** Given a query class $\mathbb{Q} \subseteq \mathsf{CQ}$, the *select-full-project equivalence* problem for $\mathbb{Q}$, denoted SelProjEquiv($\mathbb{Q}$), asks, upon input of a Boolean query $Q \in \mathbb{Q}$ and a query $Q' \in \mathbb{Q}$ whether there is a selection σ such that Q is equivalent to $(\pi_\emptyset \circ \sigma)(Q')$ where $\pi_\emptyset$ denotes the projection to the empty set.

The selection σ is reflected[9] in application α of the hardness proof and the projection in the fact that set $\mathcal{A} = \text{body}(Q)$ has no bridge variables. Hence, the following holds.

▶ **Corollary 5.11.** SelProjEquiv(ACQ) *and* SelProjEquiv(HCQ) *are* NP*-hard.*[10]

[9] An application can be decomposed into a selection which unifies head variables and an injective renaming of variables.

[10] We note that the restriction of the select-full-project equivalence problem to Boolean queries Q' is just the equivalence problem for Boolean queries which is in P for acyclic queries [10].

Surprisingly, the select-full-project equivalence problem is NP-hard not only for hierarchical but even for q-hierarchical queries – while the rewriting problem for q-hierarchical views is in P (Corollary 7.4).

▶ **Corollary 5.12.** SelProjEquiv(QHCQ) *is* NP*-hard, even over database schemas with fixed arity.*

The proof idea is explained in the full version [15].

6 A Tractable Case

In this section, we first show that the acyclic rewriting problem becomes tractable for free-connex acyclic views and acyclic queries over database schemas of bounded arity. We then define a slightly larger class of views, for which this statement holds as well. For the first result, we mainly show that rewritability with respect to a set $\mathcal{V}$ of free-connex acyclic views can be reduced to rewritability with respect to a set $\mathcal{W}$ of views of bounded arity.

▶ **Proposition 6.1.** *There is a polynomial-time algorithm that computes from each set $\mathcal{V}$ of free-connex acyclic views a set $\mathcal{W}$ of acyclic*[11] *views such that*
(a) *the arity of the views of $\mathcal{W}$ is bounded by the arity of the underlying schema, and*
(b) *every conjunctive query Q is $\mathcal{V}$-rewritable if and only if it is $\mathcal{W}$-rewritable.*
Furthermore, given a $\mathcal{W}$-rewriting of Q, a $\mathcal{V}$-rewriting of Q can be computed in polynomial time.

The proof splits each view V into a set of views obtained by the subtrees of the root $\text{head}(V)$ of a join tree of $\text{body}(V) \cup \{\text{head}(V)\}$. The arities of the children of $\text{head}(V)$ yield the desired arity bound. It can be found in the full version [15]. The main result of this section is now a simple corollary to Proposition 6.1.

▶ **Theorem 6.2.** *For every fixed k,* $\text{Rewr}^k(\mathsf{CCQ}, \mathsf{ACQ}, \mathsf{ACQ})$ *is in polynomial time and an acyclic rewriting can be computed in polynomial time, if it exists.*

Proof. Let $Q \in \mathsf{ACQ}$ be an acyclic query and $\mathcal{V} \subseteq \mathsf{CCQ}$ be a set of free-connex acyclic views over a schema $\mathcal{S}$ with arity at most k. Thanks to Proposition 6.1, from $\mathcal{V}$ an equivalent set $\mathcal{W}$ of acyclic views of arity at most k can be computed in polynomial time. The statements of the theorem thus follow immediately from Corollary 5.5. ◀

We leave the complexity of $\text{Rewr}(\mathsf{CCQ}, \mathsf{ACQ}, \mathsf{ACQ})$ as an open problem.

A closer inspection of the proof of Proposition 6.1 reveals that it does not exactly require that the views are free-connex acyclic. In fact, it suffices that each view has a partition $\mathcal{B}_1 \uplus \cdots \uplus \mathcal{B}_m$ of its body that obeys Conditions (1) and (2) below. Therefore, we turn these two requirements into a new notion. We formulate and study this notion for conjunctive queries Q, but we emphasise that we will use it for queries that define views only.

▶ **Definition 6.3.** The *weak head arity* of a query Q is the smallest k for which there is a partition $\mathcal{B}_1 \uplus \cdots \uplus \mathcal{B}_n$ of $\text{body}(Q)$ such that, for every $i \in \{1, \ldots, n\}$ and every $j < i$,
(1) $|\text{vars}(\mathcal{B}_i) \cap \text{vars}(\text{head}(Q))| \leq k$, and
(2) $\text{vars}(\mathcal{B}_i) \cap \text{vars}(\mathcal{B}_j) \subseteq \text{head}(Q)$.

[11] In fact, the views in $\mathcal{W}$ are even free-connex acyclic again. However, we do not claim it here, since it is not needed in the following, and it does not need to hold for the subsequent generalisation.

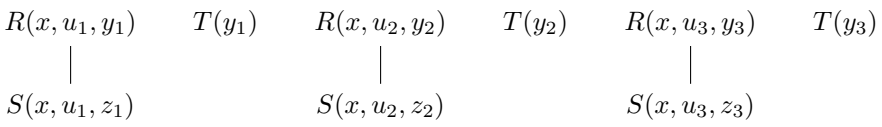


Figure 2 The cover graph $G(V_3)$ of the query V_n for $n = 3$ defined in Example 6.4.

From the proof of Proposition 6.1 it follows that every free-connex acyclic query over a fixed schema has bounded weak head arity. The following example illustrates that there are indeed views over a fixed schema that have bounded weak head arity but are not free-connex acyclic.

▶ **Example 6.4.** Let us consider the family $(V_n)_{n\in\mathbb{N}}$ of views with
- head $V_n(x, y_1, \ldots, y_n, z_1, \ldots, z_n)$, and
- body $\{R(x, u_i, y_i), S(x, u_i, z_i), T(y_i) \mid 1 \leq i \leq n\}$.

For $n \geq 1$ the view V_n is acyclic but *not* free-connex acyclic. It has, however, weak head arity 3. This is witnessed by the sets $\mathcal{B}_i = \{R(x, u_i, y_i), S(x, u_i, z_i), T(y_i)\}$ for $1 \leq i \leq n$ which form a partition satisfying the conditions of Definition 6.3.

To generalise Theorem 6.2 for views of bounded weak head arity, we thus only need to show that partitions obeying Conditions (1) and (2) of Definition 6.3 can be efficiently computed. In the remainder of this section we design an algorithm that determines the weak head arity of a given conjunctive query Q and computes a corresponding partition.

The algorithm relies on the concept of a cover graph for Q.

▶ **Definition 6.5.** The cover graph $G(Q)$ of a conjunctive query Q is the undirected graph with node set body(Q) and edges (A, A') for atoms A and A' that share a variable that does not belong to head(Q).

▶ **Example 6.6.** The cover graph of the view V_3 from Example 6.4 is depicted in Figure 2.

The following lemma states the relationship between weak head arity and cover graph.

▶ **Lemma 6.7.** *Let Q be a conjunctive query and $\mathcal{B}_1, \ldots, \mathcal{B}_n$ the connected components of its cover graph. The weak head arity of Q is the maximal number ℓ of head variables of Q in a set $\mathcal{B}_i$. Moreover, the connected components $\mathcal{B}_1, \ldots, \mathcal{B}_n$ witness that Q has weak head arity ℓ.*

Proof. Let Q be a conjunctive query and let $\mathcal{B}'_1 \uplus \cdots \uplus \mathcal{B}'_m$ be a partition of body(Q) witnessing that Q has weak head arity k.

Since by definition of the cover graph, two different sets $\mathcal{B}_i$ and $\mathcal{B}_j$ only share head variables, $\mathcal{B}_1 \uplus \cdots \uplus \mathcal{B}_n$ is a partition of body(Q) witnessing that Q has weak head arity at most ℓ, hence it holds $k \leq \ell$.

To show that $\ell \leq k$, it suffices to show that every connected component $\mathcal{B}$ in $G(Q)$ is contained in some atom set $\mathcal{B}'_i$. Towards a contradiction we assume that there is a connected component $\mathcal{B}$ with atoms from two different subsets of the partition. Since $\mathcal{B}$ is connected there must be some $A_1, A_2 \in \mathcal{B}$, connected by an edge, such that $A_1 \in \mathcal{B}'_i$ and $A_2 \in \mathcal{B}'_j$, for some $i \neq j$. By definition of $G(Q)$, A_1 and A_2 share a variable that is not part of the head of Q. But that contradicts Condition (2) in Definition 6.3. Thus, we can conclude $\ell \leq k$. ◀

Lemma 6.7 offers an algorithm to compute the weak head arity of a conjunctive query Q and a witness partition for it. It simply computes the cover graph $G(Q)$ of Q and its connected components. Then the weak head arity is the maximum number of head variables

that occur in any connected component. Furthermore, the connected components form the desired partition satisfying the conditions of Definition 6.3. We thus have the following corollary.

▶ **Corollary 6.8.** *There is an algorithm that, upon input of a conjunctive query Q, computes in polynomial time the weak head arity of Q and a partition of* $\mathrm{body}(Q)$ *that witnesses it.*

By combining Corollary 6.8 with the proofs of Proposition 6.1 and Theorem 6.2, we obtain the following generalisation of Theorem 6.2.

▶ **Theorem 6.9.** *For each fixed $k \in \mathbb{N}$, the acyclic rewriting problem for acyclic queries and acyclic views with weak head arity k is in polynomial time.*

7 The Existence of Hierarchical and q-hierarchical Rewritings

In Section 5, we have shown that every *acyclic* query has an *acyclic* rewriting if it has a rewriting at all. This gives us a guarantee that there is a rewriting that has the same complexity benefits for query evaluation as the original query.

It is natural to ask whether other, stronger properties transfer in the same fashion. In this section, we consider this question for hierarchical and q-hierarchical queries.

The following example illustrates that, as for acyclic rewritings, even if a hierarchical rewriting for a hierarchical query exists, the canonical rewriting is not necessarily hierarchical.

▶ **Example 7.1.** Consider the hierarchical query $H(x, y) \leftarrow R(x), S(y), T(x), T(y)$ and the views $V_1(x, y) \leftarrow R(x), S(y)$ and $V_2(z) \leftarrow T(z)$. The canonical rewriting leads us to the non-hierarchical rewriting $H(x, y) \leftarrow V_1(x, y), V_2(x), V_2(y)$, since $\mathrm{atoms}(y)$ and $\mathrm{atoms}(z)$ are neither disjoint nor subsets of one another. However, a hierarchical rewriting exists, for instance $H(x, y) \leftarrow V_1(x, y'), V_1(x', y), V_2(x), V_2(y)$ is a hierarchical rewriting.

It turns out that Theorem 5.4 also holds for hierarchical and q-hierarchical queries.

▶ **Theorem 7.2.** *Let $\mathcal{V}$ be a set of views.*

(a) *Every hierarchical query that has some $\mathcal{V}$-rewriting also has a hierarchical $\mathcal{V}$-rewriting.*

(b) *Every q-hierarchical query that has some $\mathcal{V}$-rewriting also has a q-hierarchical $\mathcal{V}$-rewriting.*

Similarly as for Theorem 5.4, the proof of Theorem 7.2 partitions cover descriptions. However, the strategy for doing so is different here: instead of defining the partition as the connected components of an atom set with respect to a join tree of the query at hand, here the partition guaranteed by the following lemma, which is proved in the full version [15], is used.

▶ **Lemma 7.3.** *Let Q be a hierarchical query and $(\mathcal{A}, V, \alpha, \psi)$ a cover description for Q. There is a partition $\mathcal{A}_1 \uplus \cdots \uplus \mathcal{A}_n = \mathcal{A}$ such that the following conditions hold.*

(A) *Each variable $y \notin \mathrm{vars}(\alpha(\mathrm{head}(V)))$ appears in at most one set $\mathcal{A}_i$.*

(B) *Each $\mathcal{A}_i$ is*

- **(Ba)** *a singleton set or*
- **(Bb)** *there is a variable $x \notin \mathrm{vars}(\alpha(\mathrm{head}(V)))$ that appears in every atom in $\mathcal{A}_i$.*

Similarly to Theorem 5.4, Theorem 7.2 delivers good news as well as bad news. The good news is that, since $\mathsf{QHCQ} \subseteq \mathsf{CCQ}$ and $\mathsf{QHCQ} \subseteq \mathsf{HCQ} \subseteq \mathsf{ACQ}$ hold, the rewriting problem for q-hierarchical views and hierarchical queries over a fixed schema is tractable thanks to Theorems 6.2 and 7.2. The bad news is that Theorems 5.7 and 7.2 imply NP-completeness for hierarchical queries and views.

▶ **Corollary 7.4.** $\textsc{Rewr}^k(\mathsf{QHCQ}, \mathsf{HCQ}, \mathsf{HCQ})$ *and* $\textsc{Rewr}^k(\mathsf{QHCQ}, \mathsf{QHCQ}, \mathsf{QHCQ})$ *are in polynomial time for every* $k \in \mathbb{N}$.

▶ **Corollary 7.5.** $\textsc{Rewr}^k(\mathsf{HCQ}, \mathsf{HCQ}, \mathsf{HCQ})$ *is* NP*-complete for every* $k \geq 3$.

8 Related Work

We already mentioned that our notion of cover partitions is similar to various notions from the literature. We mention three of them here. A detailed discussion for them is given in the full version [15].

- In [2], algorithms for finding rewritings with a minimal number of atoms and (maximally) contained rewritings[12] are presented. For this purpose triples (S, S', h) are considered which are comparable to cover descriptions. However, the context there is maximal rewritings and no full characterisation of exact rewritability is given.
- In [17], a characterisation for $\mathcal{V}$-rewritability is employed to find rewritings efficiently. It is in terms of *tuple coverages* and a *partition condition* corresponding to cover descriptions and partitions, respectively [17, Theorem 5, Theorem 6]. However, similar to [2], only rewritings whose body is contained in the canonical rewriting are considered.
- In [28], *MiniCon descriptions (MCDs)* are used to compute (maximally) contained rewritings (which are unions of conjunctive queries, in general). They are similar to cover descriptions, but there are several differences in the technical details, as discussed the full version [15].

Minimal cover descriptions

In [2], "minimal" triples are used to construct maximally contained rewritings. In terms of cover descriptions, an analogous definition of minimality is as follows.

▶ **Definition 8.1** (Minimal Cover Descriptions)**.** A cover description $(\mathcal{A}, V, \alpha, \psi)$ for a query Q is called *minimal* if there is no partition $\mathcal{A}_1 \uplus \cdots \uplus \mathcal{A}_k = \mathcal{A}$ for $k \geq 2$ with nonempty subsets $\mathcal{A}_1, \ldots, \mathcal{A}_k$ such that cover descriptions $C_1, \ldots, C_k$ with $C_i = (\mathcal{A}_i, V, \alpha_i, \psi_i)$ for Q exist.

It seems to be obvious to exploit such minimal cover description for the constructions in Theorem 5.4 and 7.2, instead of partitioning the set $\mathcal{A}$ of a cover description "manually". However, in contrast to our constructions, this would (possibly) not result in efficient algorithms to compute an acyclic or hierarchical rewriting from an arbitrary one, since it turns out that deciding whether a cover description is minimal is coNP-hard. A proof for the following proposition is given in the full version [15].

▶ **Proposition 8.2.** *Let* Q *be a hierarchical conjunctive query. It is* coNP*-hard to decide whether a cover description* $(\mathcal{A}, Q, \alpha, \psi)$ *is minimal.*

Applications of acyclic, free-connex acyclic, hierarchical and q-hierarchical queries

It is well known that many problems are tractable for acyclic conjunctive queries but (presumably) not for conjunctive queries in general. Notably, the evaluation, minimisation, and the containment problem are tractable for acyclic queries [30, 10, 16] but NP-complete for the class of conjunctive queries [9].

[12] In a nutshell, query Q' is a *contained rewriting* of query Q if $Q'(\mathcal{V}(D)) \subseteq Q(D)$ for any database D.

The class of free-connex conjunctive queries played a central role in the enumeration complexity of conjunctive queries. In [3], Bagan, Durand and Grandjean showed that the result of a free-connex acyclic conjunctive query can be enumerated with constant delay after a linear time preprocessing phase. Moreover, they also showed that the result of a self-join free acyclic conjunctive query that is not free-connex can not be enumerated with constant delay after a linear time preprocessing, unless $n \times n$ matrices can be multiplied in time $O(n^2)$.

Hierarchical queries have played a central role in different contexts. On the one hand, Dalvi and Suciu [12] showed that the class of hierarchical Boolean conjunctive queries characterises precisely the Boolean CQs that can be answered in polynomial time on probabilistic databases. This has been extended by Fink and Olteanu [13] to the notion of non-Boolean queries and queries with negation. On the other hand, Koutris and Suciu [24] studied hierarchical join queries in the context of query evaluation on massively parallel architectures. We refer to [14] for further applications of hierarchical queries.

The notion of q-hierarchical queries has played a central role in the evaluation of conjunctive queries under single tuple updates [6]. It is also related to factorised databases [22]. The notion of factorised databases has already been considered in various contexts [27, 4, 29]. A further recent source of information on structurally simple queries is [21].

9 Conclusion

We studied rewritability by acyclic queries or queries from CCQ, HCQ, or QHCQ. Based on a new characterisation of (exact) rewritability, we showed that acyclic queries have acyclic rewritings, if they have any CQ rewriting. The same holds for the other three query classes.

We showed that for acyclic queries and views the decision problem, whether an acyclic rewriting exists, is intractable, even for schemas with bounded arity, but becomes tractable if views have a bounded arity (even with unbounded schema arity) or are free-connex acyclic.

We leave the case of free-connex acyclic views and unbounded schemas open. Another interesting open question is the complexity of rewriting problems $\textsc{Rewr}(\mathbb{V}, \mathbb{Q}, \mathbb{R})$ with $\mathbb{R} \subsetneq \mathbb{Q}$, e.g. $\textsc{Rewr}(\mathsf{ACQ}, \mathsf{ACQ}, \mathsf{HCQ})$. Finally, it would be interesting to study whether our results can be extended to other classes of queries that can be evaluated efficiently like conjunctive queries of bounded treewidth.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

2 Foto N. Afrati and Rada Chirkova. *Answering Queries Using Views, Second Edition*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019. `doi:10.2200/S00884ED2V01Y201811DTM054`.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2007. `doi:10.1007/978-3-540-74915-8_18`.

4 Nurzhan Bakibayev, Tomás Kociský, Dan Olteanu, and Jakub Zavodny. Aggregation and ordering in factorised databases. *Proc. VLDB Endow.*, 6(14):1990–2001, 2013. `doi:10.14778/2556549.2556579`.

5 Pablo Barceló, Andreas Pieris, and Miguel Romero. Semantic optimization in tractable classes of conjunctive queries. *SIGMOD Rec.*, 46(2):5–17, 2017. `doi:10.1145/3137586.3137588`.

6 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. 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 303–318. ACM, 2017. doi:10.1145/3034786.3034789.

7 Johann Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre*. Theses, Université de Caen, April 2013. URL: https://hal.archives-ouvertes.fr/tel-01081392.

8 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. In Thomas Eiter and Leonid Libkin, editors, *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*, volume 3363 of *Lecture Notes in Computer Science*, pages 321–336. Springer, 2005. doi:10.1007/978-3-540-30570-5_22.

9 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 77–90. ACM, 1977. doi:10.1145/800105.803397.

10 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000. doi:10.1016/S0304-3975(99)00220-0.

11 Rada Chirkova and Jun Yang. Materialized views. *Found. Trends Databases*, 4(4):295–405, 2012. doi:10.1561/1900000020.

12 Nilesh N. Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 293–302. ACM, 2007. doi:10.1145/1265530.1265571.

13 Robert Fink and Dan Olteanu. A dichotomy for non-repeating queries with negation in probabilistic databases. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 144–155. ACM, 2014. doi:10.1145/2594538.2594549.

14 Robert Fink and Dan Olteanu. Dichotomies for queries with negation in probabilistic databases. *ACM Trans. Database Syst.*, 41(1):4:1–4:47, 2016. doi:10.1145/2877203.

15 Gaetano Geck, Jens Keppeler, Thomas Schwentick, and Christopher Spinrath. Rewriting with acyclic queries: Mind your head, 2022. arXiv:2201.05129.

16 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001. doi:10.1145/382780.382783.

17 Gang Gou, Maxim Kormilitsin, and Rada Chirkova. Query evaluation using overlapping views: completeness and efficiency. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 37–48. ACM, 2006. doi:10.1145/1142473.1142479.

18 Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001. doi:10.1007/s007780100054.

19 Xiao Hu and Ke Yi. Instance and output optimal parallel algorithms for acyclic joins. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *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 450–463. ACM, 2019. doi:10.1145/3294052.3319698.

20 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International*

Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017, pages 1259–1274. ACM, 2017. doi:10.1145/3035918.3064027.

21 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 375–392. ACM, 2020. doi:10.1145/3375395.3387646.

22 Jens Keppeler. *Answering Conjunctive Queries and FO+MOD Queries under Updates*. PhD thesis, Humboldt University of Berlin, Germany, 2020. URL: http://edoc.hu-berlin.de/18452/22264.

23 Benny Kimelfeld and Yehoshua Sagiv. Incrementally computing ordered answers of acyclic conjunctive queries. In Opher Etzion, Tsvi Kuflik, and Amihai Motro, editors, *Next Generation Information Technologies and Systems, 6th International Workshop, NGITS 2006, Kibbutz Shefayim, Israel, July 4-6, 2006, Proceedings*, volume 4032 of *Lecture Notes in Computer Science*, pages 141–152. Springer, 2006. doi:10.1007/11780991_13.

24 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 223–234. ACM, 2011. doi:10.1145/1989284.1989310.

25 Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In Mihalis Yannakakis and Serge Abiteboul, editors, *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA*, pages 95–104. ACM Press, 1995. doi:10.1145/212433.220198.

26 Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.*, 35(3):21:1–21:41, 2010. doi:10.1145/1806907.1806913.

27 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, 2015. doi:10.1145/2656335.

28 Rachel Pottinger and Alon Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001. doi:10.1007/s007780100048.

29 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 3–18. ACM, 2016. doi:10.1145/2882903.2882939.

30 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94. IEEE Computer Society, 1981.

Parallel Acyclic Joins with Canonical Edge Covers

Yufei Tao ✉
Chinese University of Hong Kong, Hong Kong

Abstract

In PODS'21, Hu presented an algorithm in the massively parallel computation (MPC) model that processes any acyclic join with an asymptotically optimal load. In this paper, we present an alternative analysis of her algorithm. The novelty of our analysis is in the revelation of a new mathematical structure – which we name *canonical edge cover* – for acyclic hypergraphs. We prove non-trivial properties for canonical edge covers that offer us a graph-theoretic perspective about why Hu's algorithm works.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms

Keywords and phrases Joins, Conjunctive Queries, MPC Algorithms, Parallel Computing

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.9

Related Version *Full Version*: `https://arxiv.org/abs/2201.03832` [22]

Funding This work was partially supported by GRF projects 142078/20 and 142034/21 from HKRGC.

1 Introduction

Massively parallel join processing has attracted considerable attention in recent years. This line of research makes two types of contributions. The first consists of algorithms that promise excellent performance. The second, more subtle, type of contributions comprises knowledge revealing *mathematical structures* in the underlying problems. The latter is a necessary side-product of the former. In general, as human beings switch to a more generic setting, their knowledge from restrictive settings often proves insufficient, which then necessitates deeper investigation into the problem characteristics. Traditional studies have focused on joins in the RAM computation model [4, 14, 16, 18, 19], a degenerated "parallel" setup having only one machine. Designing algorithms to work with any number of machines poses serious challenges and demands novel findings [3, 7, 8, 10, 12, 13, 15, 20, 21] beyond the RAM literature.

This paper will focus on *acyclic joins*, a class of joins with profound importance in database systems [1, 7–9, 11, 23]. Recently, Hu [8] developed a worst-case optimal massively parallel algorithm for acyclic joins. In the current work, we will provide an alternative, hopefully more accessible, analysis of her elegant algorithm. The real excitement from our analysis is the identification of a new mathematical structure – we call "canonical edge cover" – for acyclic hypergraphs. The structure reveals a unique characteristic of acyclic joins and is a core reason why Hu's algorithm works.

1.1 Problem Definition

Acyclic Joins. Let **att** be a set where each element is called an *attribute*. Let **dom** be another set where each element is called a *value*. We assume a total order on **dom**; if not, manually impose one by ordering the values arbitrarily.

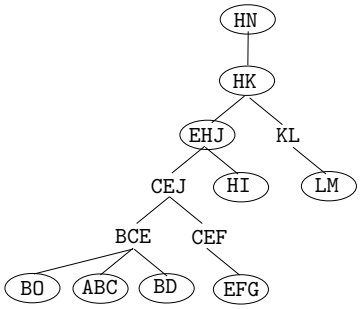


Figure 1 A hyperedge tree example.

A *tuple* over a set $U \subseteq \mathbf{att}$ is a function $\boldsymbol{u} : U \to \mathbf{dom}$. For each attribute $X \in U$, we refer to $\boldsymbol{u}(X)$ as the *value* of $\boldsymbol{u}$ on X. Given a subset $U' \subseteq U$, define $\boldsymbol{u}[U']$ – the *projection* of $\boldsymbol{u}$ on U' – as the tuple $\boldsymbol{u'}$ over U' such that $\boldsymbol{u'}(X) = \boldsymbol{u}(X)$ for every $X \in U'$. A *relation* is a set R of tuples over the same set U of attributes. We call U the *scheme* of R, a fact denoted as $scheme(R) = U$. If U is the empty set $\emptyset$, then R is also $\emptyset$.

We represent a *join query* (henceforth, simply a "join" or "query") as a set Q of relations. Define $attset(Q) = \bigcup_{R \in Q} scheme(R)$. The query result – denoted as $Join(Q)$ – is the following relation over $attset(Q)$

$$Join(Q) = \left\{ \text{tuple } \boldsymbol{u} \text{ over } attset(Q) \;\middle|\; \forall R \in Q, \boldsymbol{u}[scheme(R)] \in R \right\}.$$

If the relations in Q are $R_1, R_2, ..., R_{|Q|}$, we may represent $Join(Q)$ also as $R_1 \bowtie R_2 \bowtie ... \bowtie R_{|Q|}$.

Q can be characterized by a hypergraph $G = (V, E)$ where each vertex in V is a distinct attribute in $attset(Q)$, and each hyperedge in E is the scheme of a distinct relation in Q. E may contain identical hyperedges because two (or more) relations in Q can have the same scheme. The term "hyper" suggests that a hyperedge can have more than two attributes.

A query is *acyclic* if its hypergraph is acyclic. Specifically, a hypergraph $G = (V, E)$ is *acyclic* if we can create a tree T where

- every node in T stores (and, hence, "corresponds to") a distinct hyperedge in E;
- **(connectedness requirement)** for every attribute $X \in V$, the set S of nodes whose corresponding hyperedges contain X forms a connected subtree in T.

We will call T a *hyperedge tree* of G (also known as the *join tree* of Q in the literature).

▶ **Example 1.** Consider the hypergraph $G = (V, E)$ where $V = \{\mathtt{A}, \mathtt{B}, ..., \mathtt{O}\}$ and $E = \{\mathtt{ABC}, \mathtt{BD}, \mathtt{BO}, \mathtt{EFG}, \mathtt{BCE}, \mathtt{CEF}, \mathtt{CEJ}, \mathtt{HI}, \mathtt{LM}, \mathtt{EHJ}, \mathtt{KL}, \mathtt{HK}, \mathtt{HN}\}$. Figure 1 shows a hyperedge tree T of G. To understand the connectedness requirement, observe the connected subtree formed by the five hyperedges involving $\mathtt{E}$. ◀

As G and T both contain "vertices" and "edges", for better clarity we will obey several conventions throughout the paper. A vertex in G will always be referred to as an *attribute*, while the term *node* is reserved for the vertices in T. Furthermore, to avoid confusion with hyperedges, we will always refer to an edge in T as a *link*.

We use m to denote the *input size* of Q, defined as $\sum_{R \in Q} |R|$, namely, the total number of tuples in the relations participating in the join.

Computation Model. We assume the *massively parallel computation* (MPC) model which is popular in designing massively parallel algorithms [3, 7, 8, 10, 12, 13, 15, 20, 21]. In this model, we have p machines, each storing $\Theta(m/p)$ tuples from the relations of a query Q initially.

An algorithm executes in *rounds*, each having two phases: in the first phase, each machine performs local computation; in the second, the machines exchange messages (every message must have been generated at the end of the first phase). An algorithm must finish in a constant number of rounds, and when it does, every tuple in $Join(Q)$ must reside on at least one machine. The *load of a round* is the largest number of words received by a machine in that round. The *load of an algorithm* is the maximum load of all the rounds. The objective is to design an algorithm with the smallest load.

Math Conventions. The number p of machines is considered to be at most $m^{1-\epsilon}$, for some arbitrarily small constant $\epsilon > 0$. Every value in **dom** can be represented with $O(1)$ words. Our discussion focuses on *data complexities*, namely, we are interested in the influence of m on algorithm performance. For that reason, we assume that the hypergraph G of Q has $O(1)$ vertices. Given an integer $x \geq 1$, the notation $[x]$ represents the set $\{1, 2, ..., x\}$.

1.2 Previous Results

Fractional Edge Coverings and the AGM bound. Consider a query Q (which may or may not be acyclic) with hypergraph $G = (V, E)$. Associate every hyperedge $e \in E$ with a real-valued *weight* w_e, which falls between 0 and 1. Impose a constraint on every attribute $X \in V$: $\sum_{e \in E: X \in e} w_e \geq 1$, i.e., the total weight of all the hyperedges covering X must be at least 1. A set of weights $\{w_e \mid e \in E\}$ fulfilling all the constraints is a *fractional edge covering* of G. If we define $\sum_{e \in E} w_e$ as the *total weight* of the fractional edge covering, the *fractional edge covering number* of G – denoted as ρ – is the minimum total weight of all possible fractional edge coverings. A fractional edge covering is *optimal* if its total weight equals ρ.

The *AGM bound*, proved by Atserias, Grohe, and Marx [5], states that the size of $Join(Q)$ is always bounded by $O(m^\rho)$; recall that m is the input size of Q. Furthermore, the bound is tight: in the worst case, $|Join(Q)|$ can indeed reach $\Omega(m^\rho)$ [5].

Simplification for Acyclic Queries: Edge Covers. When Q is acyclic, $G = (V, E)$ always admits an optimal fractional edge covering with *integral* weights [8]. Recall that all the weights w_e ($e \in E$) must fall between 0 and 1. Hence, every weight in an optimal fractional edge covering must be either 0 or 1. This pleasant property allows the reader to connect ρ to edge "covers". A subset $S \subseteq E$ is an *edge cover*[1] of G if every attribute of V appears in at least one hyperedge of S. Thus, the value of ρ is simply the minimum size of all edge covers, namely, the smallest number of hyperedges that we must pick to cover all the attributes.

Join Algorithms in RAM. An algorithm able to answer Q using $O(m^\rho)$ time in the RAM model is worst-case optimal. Indeed, as $|Join(Q)|$ can be $\Omega(m^\rho)$, we need $\Theta(m^\rho)$ time just to output $Join(Q)$ in the worst case. Ngo et al. [17] designed the first algorithm that guarantees a running time of $O(m^\rho)$ for all queries. Since then, the community has discovered more algorithms [4,14,16,18,19] that are all worst-cast optimal (sometimes up to a polylogarithmic factor) but differ in their own features. For an acyclic Q, an algorithm due to Yannakakis [23] achieves a stronger sense of optimality: his algorithm runs in $O(m + |Join(Q)|)$ time, which is clearly the best regardless of $|Join(Q)|$.

[1] In case the reader is wondering, the literature uses the words "covering" and "cover" exactly the way they are used in our paper.

Join Algorithms in MPC. Koutris, Beame, and Suciu [15] showed that, in the MPC model, the AGM bound implies a worst-case lower bound of $\Omega(m/p^{1/\rho})$ on the load of any algorithm that answers a query Q, where m is the input size of Q and ρ is the fractional edge covering number of the hypergraph $G = (V, E)$ defined by Q.

The above negative result has motivated considerable research looking for MPC algorithms whose loads are bounded by $O(m/p^{1/\rho})$, ignoring polylogarithmic factors; such algorithms are worst-case optimal. The goal has been realized only on four query classes. The first consists of all the cartesian-product queries (i.e., the relations in Q have disjoint schemes); see [3,6,13] for several optimal algorithms on such queries. The second is the so-called Loomis-Whitney join, where E consists of all the $|V|$ possible hyperedges of $|V| - 1$ attributes; see [15] for an optimal algorithm for such joins. The third class includes every query where all the hyperedges in G contain *at most* two attributes; see [12,13,21] for the optimal algorithms. The fourth class comprises all the acyclic queries, which were recently solved by Hu [8] optimally. It is worth pointing out that Hu's algorithm subsumes an earlier algorithm of [9] which is worst-case optimal on a subclass of acyclic queries.

Although it still remains elusive what other query classes can be settled with load $O(m/p^{1/\rho})$, now we know that this is *unachievable* for certain queries. In [8], Hu constructed a class of queries for which every algorithm must incur a load of $\omega(m/p^{1/\rho})$ in the worst case. The result of [8] suggests that additional parameters – other than m, p, and ρ – are needed to describe the worst-case optimality of an ideal MPC algorithm. We will not delve into the issue further because it does not apply to acyclic queries (the focus of this paper), but the reader may consult the recent works [8,20] for the latest development on that issue. Finally, we remark that several algorithms [2,9,10] are able to achieve a load sensitive to the join size $|Join(Q)|$.

1.3 Our Contributions

The first, easy-to-discern, contribution of our paper is a new analysis of Hu's algorithm [8] for acyclic queries. Our second contribution is the introduction of *canonical edge cover* as a mathematical structure inherent in acyclic queries. We prove a suite of graph-theoretic properties for canonical edge covers and use them to give a more fundamental interpretation of the design choices in Hu's algorithm. The rest of the section will provide an overview of our results and techniques.

Clustering, k-Groups, k-Products, and Induced Loads. We first create a conceptual framework to state Hu's and our results on a common ground. Define a *clustering* of E (the hyperedge set of G) as a set $\{E_1, E_2, ..., E_s\}$ for some $s \geq 1$ where (i) each E_i is a subset of E, $i \in [s]$, and (ii) $\bigcup_i E_i = E$. We call each E_i a *cluster*; note that the clusters need *not* be disjoint.

Fix an arbitrary clustering $C = \{E_1, E_2, ..., E_s\}$. Given an integer $k \geq 1$, define a *k-group* of C as a collection of k hyperedges, each taken from a distinct cluster.

▶ **Example 2.** Let $G = (V, E)$ be the hypergraph in Example 1 (Figure 1). $C = \{\{$`BO`, `BCE`, `CEJ`$\}$, $\{$`ABC`, `BCE`, `CEJ`$\}$, $\{$`BD`, `BCE`, `CEJ`$\}$, $\{$`EFG`, `CEF`, `CEJ`$\}$, $\{$`HI`$\}$, $\{$`EHJ`$\}$, $\{$`LM`, `KL`$\}$, $\{$`HK`$\}$, $\{$`HN`$\}\}$ is a clustering of E. A 3-group example is $\{$`ABC`, `BD`, `EFG`$\}$. Note that the hyperedges in a k-group need not be distinct. For example, $\{$`CEJ`, `CEJ`, `CEJ`$\}$ is also a 3-group: the first `CEJ` is taken from the cluster $\{$`ABC`, `BCE`, `CEJ`$\}$, the second from $\{$`BD`, `BCE`, `CEJ`$\}$, and the third from $\{$`EFG`, `CEF`, `CEJ`$\}$. For a non-example, $\{$`ABC`, `LM`, `KL`$\}$ is not a 3-group. ⌟

For each hyperedge $e \in E$, let $R(e)$ represent the relation in Q whose scheme is e. Given a k-group K of the clustering C, we define the *Q-product* of K as $\prod_{e \in K} |R(e)|$ (i.e., the cartesian-product size of all the relevant relations). Given an integer k, we define the *max (k, Q)-product* of C – denoted as $P_k(Q, C)$ – as the maximum Q-product of all the k-groups of C.

▶ **Example 3.** Continuing on the previous example, the Q-product of the 3-group $\{\texttt{ABC}, \texttt{BCE}, \texttt{CEJ}\}$ is $|R(\texttt{ABC})| \cdot |R(\texttt{BCE})| \cdot |R(\texttt{CEJ})|$, while that of the 3-group $\{\texttt{CEJ}, \texttt{CEJ}, \texttt{CEJ}\}$ is $|R(\texttt{CEJ})|^3$. ⌟

Define the *Q-induced load* of C as

$$\max_{k=1}^{s} (P_k(Q, C)/p)^{1/k} \tag{1}$$

As $P_k(Q, C) \leq m^k$ for any $k \in [s]$, it must hold that $(1) \leq m/p^{1/s}$.

We can now give a more detailed account of Hu's result [8]. She proved that the load of her algorithm is bounded by $O(L)$, where L is the Q-induced load of a *certain* clustering with size $s = \rho$, and ρ is the fractional edge covering number of G. It thus follows immediately that $L \leq m/p^{1/s}$. In [8], Hu presented a recursive procedure to identify the clustering C whose Q-induced load equals the target L. The procedure, however, is somewhat sophisticated, making it difficult to describe the target C in a succinct manner. Such difficulty is unjustified, especially given the algorithm's elegance, and indicates the existence of a hidden mathematical structure.

Our Results and Techniques. A hypergraph G can have many optimal edge covers (all of which must have size ρ). While Hu's analysis [8] assumes an arbitrary optimal edge cover, we will be choosy about what we work with. In Figure 1, the 9 circled nodes constitute a *canonical edge cover* F of G. Let us give an informal but intuitive explanation of how to construct this F. After rooting the tree in Figure 1 at HN, we add to F all the leaf nodes: BO, ABC, BD, EFG, HI, LM. Then, we process the non-leaf nodes bottom up. In processing BCE, we ask: which attributes will disappear as we ascend further in the tree? The answer is B, which is thus a "disappearing" attribute of BCE. Then, we ask: does F already cover B? The answer is yes, due to the existence of BO; we therefore do *not* include BCE in F. We process BCE, CEF, and CEJ similarly, none of which enters F. At EHJ, we find disappearing attributes E and J. In general, as long as one disappearing attribute has not been covered by F, we pick the node; this is why EHJ is in F. The other nodes HK and HN in F are chosen based on the same reasoning.

We show that a canonical edge cover determined this way has appealing properties which fit the recursive strategy behind Hu's algorithm very well. At a high level, Hu's algorithm works by simplifying G into a number of "residual" hypergraphs to be processed recursively. Interestingly, with trivial modifications (such as removing the attributes that have become irrelevant), a canonical edge cover of G *remains canonical on every residual hypergraph*. This is the most crucial property we utilize to relate the load of the original query to those of the "residual queries" in forming up a working recurrence.

Our techniques also provide a simple and natural way to pinpoint a clustering C that can be used to bound the algorithm's load. Consider the canonical edge cover F shown in Figure 1 (the circled nodes). For each node in F, take a "signature path" by walking up and stopping right before reaching its lowest proper ancestor in F. For example, the signature path of ABC is {ABC, BCE, CEJ} (note: the path does not contain EHJ). Likewise, the signature path of LM is {LM, KL}. The signature paths of all the nodes in F together

produce the clustering C given in Example 2. Our main result (Theorem 22) states that the Q-induced load of C is an upper bound on the load of Hu's algorithm. Because C has a size at most ρ, the algorithm's load is thus bounded by $O(m/p^{1/\rho})$.

2 Canonical Edge Covers for Acyclic Hypergraphs

This section is purely graph theoretic: we will establish several new properties for acyclic hypergraphs. Let $G = (V, E)$ be an acyclic hypergraph. A hyperedge $e_1 \in E$ is *subsumed* if it is a subset of another hyperedge $e_2 \in E$, i.e., $e_1 \subseteq e_2$. If an attribute X appears in only a single hyperedge, we call X an *exclusive attribute*; otherwise, X is *non-exclusive*. Unless otherwise stated, we allow G to be an arbitrary acyclic hypergraph. In particular, this means that E can contain two or more hyperedges with the same attributes (nonetheless, they are still distinct hyperedges) and may even have empty hyperedges (i.e., with no attributes at all). G is *clean* if E has no subsumed edges. Some of our results will apply only to clean hypergraphs.

Denote by T a hyperedge tree of G (the existence of T is guaranteed; see Section 1.1). By rooting T at an arbitrary leaf, we can regard T as a *rooted* tree. Make all the links[2] of T point *downwards*, i.e., from parent to child. This way, T becomes a directed acyclic graph.

Now that there are two views of T (i.e., undirected and directed), we ought to be careful with terminology. By default, we will treat T as a directed tree. Accordingly, a *leaf* of T is a node with out-degree 0, a *path* is a sequence of nodes where each node has a link pointing to the next node, and a *subtree* rooted at a node e is the directed tree induced by the nodes reachable from e in T. Sometimes, we may revert back to the undirected view of T. In that case, we use the term *raw leaf* for a leaf in the undirected T (i.e., a raw leaf can be a leaf or the root under the directed view)

2.1 Fundamental Definitions and Properties

Summits and Disappearing Attributes. We say that the root of T is the *highest* node in T and, in general, a node is *higher* (or *lower*) than any of its proper descendants (or ancestors). For each attribute $X \in V$, we define the *summit* of X as the highest node (a.k.a. a hyperedge) that contains X. If node e is the summit of X, we call X a *disappearing* attribute in e. By acyclicity's connectedness requirement (Section 1.1), X can appear only in the subtree rooted at e and hence "disappears" as soon as we leave the subtree.

▶ **Example 4.** Let $G = (V, E)$ be the hypergraph in Example 1 whose (rooted) hypergraph tree T is shown in Figure 1. The summit of `C` is node `CEJ`. Thus, `C` is a disappearing attribute of `CEJ`. Node `EHJ` is the summit of `E` and `J`. Hence, both `E` and `J` are disappearing attributes of `EHJ`. ⌟

Canonical Edge Cover. We say that a subset $S \subseteq E$ *covers* an attribute $X \in V$ if S has a hyperedge containing X. Recall that an optimal edge cover of G is the smallest S covering every attribute in V. Optimal edge covers are not unique. Some are of particular importance to us; and we will identify them as "canonical". Towards a procedural definition, consider the following algorithm:

[2] Remember that we refrain from saying "edges" of T; see Section 1.1.

edge-cover (T) /* T is rooted */
1. $F_{\text{tmp}} = \emptyset$
2. obtain a reverse topological order $e_1, e_2, ..., e_{|E|}$ of the nodes (i.e., hyperedges) in T
3. **for** $i = 1$ to $|E|$ **do**
4. **if** e_i has a disappearing attribute not covered by F_{tmp} **then** add e_i to F_{tmp}
5. **return** F_{tmp}

▶ **Lemma 5.** *The output of* edge-cover *– denoted as F – is an optimal edge cover of G, and does not depend on the reverse topological order at Line 2. Furthermore, if G is clean, F includes all the raw leaves of T.*

All the missing proofs can be found in the full version [22]. We refer to F as the *canonical edge cover* (CEC) of G induced by T. The size of F is precisely the fractional edge covering number ρ of Q.

▶ **Example 6.** Continuing on the previous example, consider the reverse topological order of T: ABC, BD, BO, BCE, EFG, CEF, CEJ, HI, EHJ, LM, KL, HK, HN. When processing ABC, edge-cover adds it to F_{tmp} because ABC has a disappearing attribute A and yet $F_{\text{tmp}} = \emptyset$. When processing BCE, $F_{\text{tmp}} = \{\text{ABC}, \text{BD}, \text{BO}\}$. BCE has a disappearing attribute B, which, however, has been covered by F_{tmp}. Thus, B is not added to F_{tmp}. The final output of the algorithm is $F = \{\text{ABC}, \text{BD}, \text{BO}, \text{EFG}, \text{HI}, \text{LM}, \text{EHJ}, \text{HK}, \text{HN}\}$, which is the CEC of G induced by T. ⌟

Signature Paths. Whenever F includes the root of T, we can define a *signature path* – denoted as $\text{sigpath}(f, T)$ – for each node (i.e., hyperedge) $f \in F$. Specifically, $\text{sigpath}(f, T)$ is a set of nodes defined as follows:

- If f is the root of T, $\text{sigpath}(f, T) = \{f\}$.
- Otherwise, let $\hat{f}$ be the lowest node in F that is a proper ancestor of f. Then, $\text{sigpath}(f, T)$ is the set of nodes on the path from $\hat{f}$ to f, except $\hat{f}$.

▶ **Example 7.** Consider the set F obtained in the previous example. If $f = \text{HN}$, then the signature path of f is $\{\text{HN}\}$. If $f = \text{ABC}$, then $\hat{f} = \text{EHJ}$; and the signature path of f is $\{\text{ABC}, \text{BCE}, \text{CEJ}\}$. ⌟

(Clean G) Clustering, Anchor Leaf, and Anchor Attribute. Consider $G = (V, E)$ now as a clean hypergraph. Let F be the CEC of G induced by a hyperedge tree T of G. As F contains the root and leaves of T (Lemma 5), $\{\text{sigpath}(f, T) \mid f \in F\}$ is a clustering of E. If f is not the root of T, we call $\text{sigpath}(f, T)$ a *non-root* cluster.[3]

Let f° be a leaf node in F, and $\hat{f}$ be the lowest proper ancestor of f° in F. We call f° an *anchor leaf* of T if two conditions are satisfied:

- $\hat{f}$ has no non-leaf proper descendants in F.
- f° has an attribute A° such that $A^\circ \notin \hat{f}$ but $A^\circ \in e$ for every node $e \in \text{sigpath}(f^\circ, T)$.

A° will be referred to as an *anchor attribute* of f°.

▶ **Lemma 8.** *If G is clean, F always contains an anchor leaf.*

▶ **Example 9.** From the F constructed earlier, we obtain the clustering $C = \{\{\text{BO}, \text{BCE}, \text{CEJ}\}, \{\text{ABC}, \text{BCE}, \text{CEJ}\}, \{\text{BD}, \text{BCE}, \text{CEJ}\}, \{\text{EFG}, \text{CEF}, \text{CEJ}\}, \{\text{HI}\}, \{\text{EHJ}\}, \{\text{LM}, \text{KL}\}, \{\text{HK}\}, \{\text{HN}\}\}$. Other than $\{\text{HN}\}$, all the clusters in C are non-root clusters. ABC is an anchor leaf of T

[3] If f is the root of T, $\text{sigpath}(f, T)$ contains just f itself.

with an anchor attribute C. HI is another anchor leaf with an anchor attribute I. For a non-example, BD is not an anchor leaf because it does not have an attribute that exists in all the nodes in sigpath(BD, T) = {BD, BCE, CEJ}. Furthermore, LM is not an anchor leaf because HK, the lowest proper ancestor of LM in F, has a non-leaf proper descendant in F (i.e., EHJ). ⌟

2.2 (Clean G) Properties on Residual Hypergraphs

This subsection assumes $G = (V, E)$ to be clean. Let T be a hyperedge tree of G and F be the CEC induced by T. Fix an arbitrary anchor leaf f° of T and an anchor attribute A° of f°. We will analyze how the CEC changes as G is simplified based on f° and A°.

2.2.1 Simplification 1

The first simplification is based on removing attribute A° from G.

Residual Hypergraph. Let $G' = (V', E')$ be the *residual hypergraph* obtained by eliminating A° from G: $V' = V \setminus \{A^\circ\}$, and E' collects a hyperedge $e' = e \setminus \{A^\circ\}$ for every $e \in E$.[4] We characterize the one-one correspondence between E' and E by introducing a function $map(e) = e'$ and its inverse function $map^{-1}(e') = e$. Let T' be the hyperedge tree of G' obtained by discarding A° from every node in T (note: G' is not necessarily clean).

Canonical Edge Cover. Define

$$F' = \begin{cases} F \setminus \{f^\circ\} & \text{if } map(f^\circ) \text{ is subsumed in } G' \\ \{map(f) \mid f \in F\} & \text{otherwise} \end{cases} \tag{2}$$

▶ **Example 10.** Continuing on the previous example, if we choose f° = ABC with A° = C and eliminate C from the tree T in Figure 1, we obtain the hyperedge tree T' in Figure 2a, where the circled nodes constitute the set F'. Similarly, if we choose f° = HI with A° = I, then T' and F' are as demonstrated in Figure 2b. ⌟

▶ **Lemma 11.** *If G is clean, F' is the CEC of G' induced by T'. Furthermore, if $map(f^\circ)$ is subsumed in G', then A° must be an exclusive attribute in f°.*

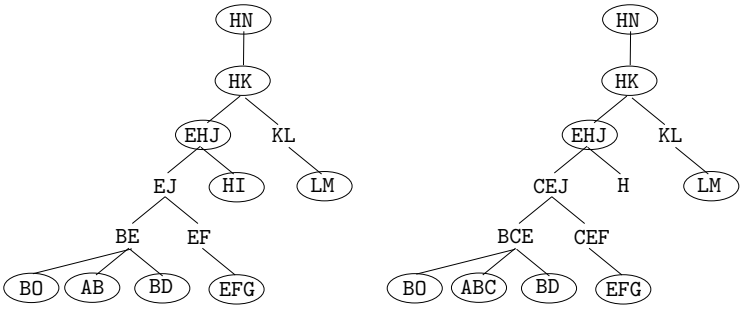
As a corollary, if $map(f^\circ)$ is subsumed in G', then every hyperedge of G, except f°, is directly retained in G'; furthermore, $map(f^\circ)$ is the only subsumed edge in G'. The next lemma gives another property of F' that holds no matter if G is clean.

▶ **Lemma 12.** *If a hyperedge e' of G' is subsumed, then $e' \notin F'$*

Cleansing. Even though G is clean, the residual hypergraph G' may contain subsumed hyperedges. Next, we describe a *cleansing* procedure which converts G' into a clean hypergraph $G^* = (V', E^*)$ (note that G^* has the same vertices as G') and converts T' into a rooted hyperedge tree T^* of G^*.

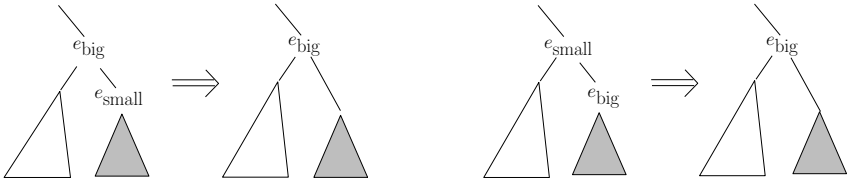
Cleansing is simple if $map(f^\circ)$ is subsumed in G'. In this case, G^* is the hypergraph obtained by removing $map(f^\circ)$ from G', and T^* is the tree obtained by removing the leaf $map(f^\circ)$ from T'. If $map(f^\circ)$ is not subsumed, the cleansing algorithm is:

[4] If $e = \{A^\circ\}$, E' collects $e' = \emptyset$.



(a) T' and F' after removing C. (b) T' and F' after removing I.

Figure 2 Residual hypergraphs.



(a) e_{big} parents e_{small}. (b) e_{small} parents e_{big}.

Figure 3 Two cases of cleansing.

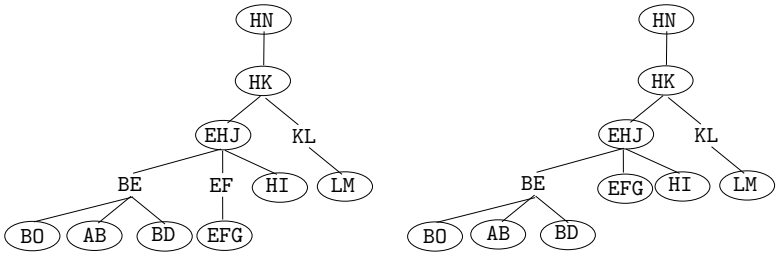
cleanse (G', T') /* condition: $map(f^\circ)$ not subsumed */
1. $G^* = G', T^* = T'$
2. **while** G^* has hyperedges e_{small} and e_{big} such that $e_{small} \subseteq e_{big}$ and they are connected by a link in T^* **do**
3. remove e_{small} from G^* and T^*
   /* $e_{small} \notin F'$ by Lemma 12 */
4. **if** e_{big} was the parent of e_{small} in T^* **then**
5. make e_{big} the new parent for all the child nodes of e_{small}; see Figure 3a
   else
6. make e_{big} the new parent for the child nodes of e_{small}, and
   make e_{big} a child of the (original) parent of e_{small} in T^*; see Figure 3b
7. **return** G^* and T^*

At the end of cleansing, we always set $F^* = F'$, regardless of whether $map(f^\circ)$ is subsumed.

▶ **Lemma 13.** *After cleansing, F^* is the CEC of G^* induced by T^*.*

▶ **Example 14.** In Example 10, the residual hypergraph G' in Figure 2a has two subsumed hyperedges EJ and EF, each removed by an iteration of cleanse. Suppose that the first iteration sets $e_{small} =$ EJ and $e_{big} =$ EHJ (this is a case of Figure 3a). Figure 4a illustrates the T^* after removing EJ. The next iteration sets $e_{small} =$ EF and $e_{big} =$ EFG (a case of Figure 4b). Figure 4b illustrates the T^* after removing EF. In both Figure 4a and 4b, the circled nodes constitute the CEC of G^* induced by T^*. ⌟

Distinct Clusters Lemma. The next property concerns the hypergraph $G^* = (V', E^*)$ after cleansing and the original hypergraph $G = (V, E)$. Recall that T^* and T are hyperedge trees of G^* and G, respectively. Before proceeding, the reader should recall that every hyperedge $e^* \in E^*$ corresponds to a distinct hyperedge $e \in E$, which is the hyperedge given by $map^{-1}(e^*)$.



(a) After removing EJ. (b) After removing EF.

Figure 4 Simplification 1.

Consider once again the CEC F of G, i.e., the original hypergraph, induced by T. As mentioned in Section 2.1, $C = \{\text{sigpath}(f, T) \mid f \in F\}$ is a clustering of E. By the same reasoning, because F^* is the CEC of G^* induced by T^* (Lemma 13), $C^* = \{\text{sigpath}(f^*, T^*) \mid f^* \in F^*\}$ must be a clustering of E^*. The following lemma draws a connection between C and C^*:

▶ **Lemma 15** (Distinct Clusters Lemma). *For any $1 \leq k \leq |F^*|$, if $\{e_1^*, ..., e_k^*\}$ is a k-group of C^*, then $\{map^{-1}(e_1^*), ..., map^{-1}(e_k^*)\}$ is a k-group of C.*

By definition of k-group, $e_1^*, ..., e_k^*$ originate from k distinct clusters in C^*. The lemma promises k different clusters in C each containing a distinct hyperedge in $\{map^{-1}(e_1^*), ..., map^{-1}(e_k^*)\}$.

▶ **Example 16.** Consider the T^* (and hence G^*) and F^* illustrated in Figure 4b. The clustering C^* is $\{\{\texttt{AB}, \texttt{BE}\}, \{\texttt{BO}, \texttt{BE}\}, \{\texttt{BD}, \texttt{BE}\}, \{\texttt{EFG}\}, \{\texttt{EHJ}\}, \{\texttt{HI}\}, \{\texttt{LM}, \texttt{KL}\}, \{\texttt{HK}\}, \{\texttt{HN}\}\}$. Because $\{\texttt{BE}, \texttt{EFG}, \texttt{KL}\}$ is a 3-group of C^*, Lemma 15 asserts that $\{map^{-1}(\texttt{BE}), map^{-1}(\texttt{EFG}), map^{-1}(\texttt{KL})\} = \{\texttt{BCE}, \texttt{EFG}, \texttt{KL}\}$ must be a 3-group of the clustering C in Example 9. ⌟

2.2.2 Simplification 2

The second simplification decomposes G into multiple hypergraphs based on $\text{sigpath}(f^\circ, T)$.

Decomposition. Define Z to be the set of nodes z in T satisfying: z is not in $\text{sigpath}(f^\circ, T)$ but the parent of z is. For each $z \in Z$, define a rooted tree T_z^* as follows:

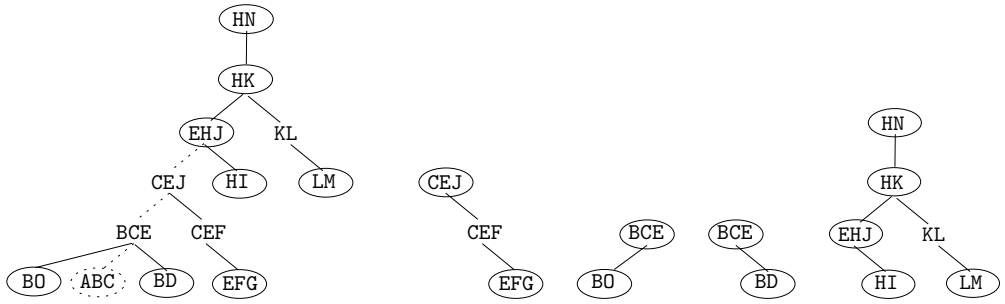
- The root of T_z^* is the parent of z in T.
- The root of T_z^* has only one child in T_z^*, which is z.
- The subtree rooted at z in T_z^* is the same as the subtree rooted at z in T.

Separately, define $\bar{T}^*$ as the rooted tree obtained by removing from T the subtree rooted at the highest node in $\text{sigpath}(f^\circ, T)$.

From each T_z^*, generate a hypergraph $G_z^* = (V_z^*, E_z^*)$. Specifically, E_z^* includes all and only the nodes (each being a hyperedge) in T_z^*, and V_z^* is the set of attributes appearing in at least one hyperedge in E_z^*. Likewise, from $\bar{T}^*$, generate a hypergraph $\bar{G}^* = (\bar{V}^*, \bar{E}^*)$ where $\bar{E}^*$ includes all and only the nodes in $\bar{T}^*$, and $\bar{V}^*$ is the set of attributes appearing in at least one hyperedge in $\bar{E}^*$.

Because G is clean, so must be all the generated hypergraphs. Furthermore, each of them has fewer edges than G.[5] For each $z \in Z$, T_z^* is a hyperedge tree of G_z^*; similarly, $\bar{T}^*$ is a hyperedge tree of $\bar{G}^*$.

[5] Because f° does not appear in any of the generated hypergraphs.



(a) The dotted path is sigpath(ABC, T). (b) $T^*_{\texttt{CEF}}$ (c) $T^*_{\texttt{BO}}$ (d) $T^*_{\texttt{BD}}$ (e) $\bar{T}^*$

Figure 5 Decomposition.

▶ **Example 17.** In our running example, $f^\circ = \texttt{ABC}$, whose signature path is sigpath$(f^\circ, T) = \{\texttt{ABC}, \texttt{BCE}, \texttt{CEJ}\}$; see Figure 5a. $Z = \{\texttt{BO}, \texttt{BD}, \texttt{CEF}\}$. Figure 5b, 5c, and 5d illustrate T^*_z for $z = \texttt{CEF}$, BO, and BD, respectively. Figure 5e gives $\bar{T}^*_z$. ⌟

Canonical Edge Covers. Recall that F is the CEC of G induced by T. Next, we derive the CECs of the hypergraphs generated from the decomposition. For each $z \in Z$, define

$$F^*_z = \{\text{parent of } z\} \cup (F \cap E^*_z). \tag{3}$$

Also, define

$$\bar{F}^* = F \cap \bar{E}^*. \tag{4}$$

▶ **Lemma 18.** *For each node $z \in Z$, F^*_z is the CEC of G^*_z induced by T^*_z. Furthermore, $\bar{F}^*$ is the CEC of $\bar{G}^*$ induced by $\bar{T}^*$.*

▶ **Example 19.** We have circled the nodes in F^*_z in Figure 5b, 5c, and 5d for $z = \texttt{CEF}$, BO, and BD, respectively. Similarly, the circled nodes in Figure 5e constitute $\bar{F}^*_z$. ⌟

Distinct Clusters Lemma 2. We close the section with a property resembling Lemma 15.

Consider any $z \in Z$. Because $G^*_z = (V^*_z, E^*_z)$ is clean and F^*_z is the CEC of G^*_z induced by T^*_z, $C^*_z = \{\text{sigpath}(f^*, T^*_z) \mid f^* \in F^*_z\}$ is a clustering of E^*_z. Similarly, regarding $\bar{G}^* = (\bar{V}^*, \bar{E}^*)$, $\bar{C}^* = \{\text{sigpath}(f^*, \bar{T}^*) \mid f^* \in \bar{F}^*\}$ is a clustering of $\bar{E}^*$.

Define a *super-k-group* to be a set of hyperedges $K = \{e_1, e_2, ..., e_k\}$ satisfying:

- Each e_i, $i \in [k]$, is taken from a cluster of $\bar{C}^*$ or a non-root cluster[6] of C^*_z for some $z \in Z$.
- No two hyperedges in K are taken from the same cluster.

Before delving into the next lemma, the reader should recall that $\{\text{sigpath}(f, T) \mid f \in F\}$ is a clustering of E.

▶ **Lemma 20** (Distinct Clusters Lemma 2)**.** *If $\{e_1, e_2, ..., e_k\}$ is a super-k-group, then $\{e_1, e_2, ..., e_k\}$ must be a k-group of the clustering $\{\text{sigpath}(f, T) \mid f \in F\}$.*

▶ **Example 21.** In Figure 5, $C^*_{\texttt{CEF}} = \{\{\texttt{EFG}, \texttt{CEF}\}, \{\texttt{CEJ}\}\}$, $C^*_{\texttt{BO}} = \{\{\texttt{BO}\}, \{\texttt{BCE}\}\}$, $C^*_{\texttt{BD}} = \{\{\texttt{BD}\}, \{\texttt{BCE}\}\}$, $\bar{C}^* = \{\{\texttt{HI}\}, \{\texttt{EHJ}\}, \{\texttt{HK}\}, \{\texttt{HN}\}, \{\texttt{LM}, \texttt{KL}\}\}$. A super-4-group is {CEF, BO, BD, KL}. Lemma 20 assures us that {CEF, BO, BD, KL} must be a 4-group in the clustering C given in Example 9. ⌟

[6] Namely, e_i cannot be the root of T^*_z.

3 An MPC Algorithm

The rest of the paper will apply the theory of CECs to solve acyclic queries in the MPC model. We will describe a variant of Hu's algorithm [8] in this section[7] and present our analysis in the next section. Denote by Q the acyclic query to be answered. Let $G = (V, E)$ be the hypergraph of Q. We assume G to be clean; otherwise, Q can be converted to a clean query having the same result with load $O(m/p)$ [8]. We will also assume that Q has at least two relations; otherwise, the query is trivial and requires no communication.

3.1 Configurations

Let T be a hyperedge tree of G and F be the CEC of G induced by T. The size of F is precisely ρ, the fractional edge covering number of Q (Section 1.2). As explained in Section 2.1, when G is clean,

$$C = \{\text{sigpath}(f, T) \mid f \in F\} \tag{5}$$

is a clustering of E. Let f° be an anchor leaf of T and A° an anchor attribute of f° (Section 2.1); remember that A° appears in all the hyperedges of $\text{sigpath}(f^\circ, T)$. Define

$$L = \text{the } Q\text{-induced load of } C. \tag{6}$$

The reader can review Equation (1) for the definition of "Q-induced load".

For each hyperedge $e \in E$, as before $R(e)$ denotes the relation in Q corresponding to e. Fix a value $x \in \mathbf{dom}$. Given an $e \in \text{sigpath}(f^\circ, T)$, we define the *$A^\circ$-frequency* of x in $R(e)$ as the number of tuples $\boldsymbol{u} \in R(e)$ such that $\boldsymbol{u}(A^\circ) = x$. Further define the *signature-path A°-frequency* of x as the sum of its A°-frequencies in the $R(e)$ of all $e \in \text{sigpath}(f^\circ, T)$. A value $x \in \mathbf{dom}$ is

- *heavy*, if its signature-path A°-frequency is at least L;
- *light*, otherwise.

Divide $\mathbf{dom}$ into disjoint intervals such that the light values in each interval have a total signature-path A°-frequency of $\Theta(L)$. We will refer to those intervals as the *light intervals* of A°. The total number of heavy values and light intervals is at most

$$\sum_{e \in \text{sigpath}(f^\circ, T)} \frac{|R(e)|}{L} = O\left(\max_{e \in \text{sigpath}(f^\circ, T)} \frac{|R(e)|}{L}\right) = O\left(\frac{\max\ (1, Q)\text{-product of } C}{L}\right) = O(p) \tag{7}$$

where the first equality used the fact that $\text{sigpath}(f^\circ, T)$ has $O(1)$ edges and the second equality applied the definition of max (k, Q)-product (see Section 1.3).

A *configuration* η is either a heavy value or a light interval. Equation (7) implies that the number of configurations is $O(p)$. For each hypergraph $e \in E$, define a relation $R(e, \eta)$ as follows:

- if η is a heavy value, $R(e, \eta)$ includes all and only the tuples $\boldsymbol{u} \in R(e)$ satisfying $\boldsymbol{u}(A^\circ) = \eta$;
- if η is a light interval, $R(e, \eta)$ includes all and only the tuples $\boldsymbol{u} \in R(e)$ where $\boldsymbol{u}(A^\circ)$ is a light value in η.

[7] Our algorithm follows Hu's ideas [8] but differs in certain details. For example, Hu's algorithm takes an arbitrary optimal edge cover of G as the input, while we insist on working with a CEC.

Note that $R(e, \eta) = R(e)$ if $A^\circ \notin e$. Let Q_η be the query defined by $\{R(e, \eta) \mid e \in E\}$. Our objective is to compute $Join(Q_\eta)$ for all η in parallel. The final result $Join(Q)$ is simply $\bigcup_\eta Join(Q_\eta)$.

The rest of the section will explain how to solve $Join(Q_\eta)$ for an arbitrary η. We allocate

$$p_\eta = \Theta\left(1 + \max_{k=1}^{|F|} \frac{P_k(Q_\eta, C)}{L^k}\right) \tag{8}$$

machines for this purpose, where $P_k(Q_\eta, C)$ is the max (k, Q_η)-product of C.

3.2 Solving Q_η When η is a Heavy Value

Define the residual hypergraph $G' = (V', E')$ after removing A°, and also functions $map(.)$ and $map^{-1}(.)$ as in Section 2.2.1. We compute $Join(Q_\eta)$ in five steps.

Step 1. Send the tuples of $R(e, \eta)$, for all $e \in E$, to the p_η allocated machines such that each machine receives $\Theta(\frac{1}{p_\eta} \sum_{e \in E} |R(e, \eta)|)$ tuples.

Step 2. For each $e \in E$, convert $R(e, \eta)$ to $R^*(e', \eta)$ where $e' = map(e) = e \setminus \{A^\circ\}$. Specifically, $R^*(e', \eta)$ is a copy of $R(e, \eta)$ but with A° discarded, or formally, $R^*(e', \eta) = \{\boldsymbol{u}[e'] \mid \text{tuple } \boldsymbol{u} \in R(e, \eta)\}$. No communication occurs as each machine simply discards A° from every tuple $\boldsymbol{u} \in R(e, \eta)$ in the local storage.

Step 3. Cleanse G' into $G^* = (V', E^*)$. As explained in Section 2.2.1, this may or may not require calling algorithm cleanse. If called, cleanse identifies in each iteration two hyperedges e_{small} and e_{big} in the current G^* and removes e_{small}. Accordingly, we perform a *semi-join* between $R^*(e_{\text{small}}, \eta)$ and $R^*(e_{\text{big}}, \eta)$, which removes every tuple $\boldsymbol{u}$ from $R^*(e_{\text{big}}, \eta)$ with the property that $\boldsymbol{u}[e_{\text{small}}]$ is absent from $R^*(e_{\text{small}}, \eta)$. $R^*(e_{\text{small}}, \eta)$ is discarded after the semi-join.

Step 4. Let Q^*_η be the query defined by the relation set $\{R^*(e^*, \eta) \mid e^* \in E^*\}$. Compute $Join(Q^*_\eta)$ using p_η machines recursively. Note that the number of participating attributes has decreased by 1 for the recursion.

Step 5. We output $Join(Q_\eta)$ by augmenting each tuple $\boldsymbol{u} \in Join(Q^*_\eta)$ with $\boldsymbol{u}(A^\circ) = \eta$. No communication is needed.

3.3 Solving Q_η When η is a Light Interval

Define Z, $G^*_z = (V^*_z, E^*_z)$ (for each $z \in Z$), C^*_z, $\bar{G}^* = (\bar{V}^*, \bar{E}^*)$, and $\bar{C}^*$ all in the way described in Section 2.2.2. We compute $Join(Q_\eta)$ in four steps.

Step 1. Same as Step 1 of the algorithm in Section 3.2.

Step 2. For each $e \in \text{sigpath}(f^\circ, T)$, broadcast $R(e, \eta)$ to all p_η machines. By definition of light interval, the size of $R(e, \eta)$ is at most L.

Step 3. For each $z \in Z$, define a query $Q^*_{\eta,z} = \{R(e, \eta) \mid e \in E^*_z\}$. Similarly, for $\bar{G}^*$, define a query $\bar{Q}^*_\eta = \{R(e, \eta) \mid e \in \bar{E}^*\}$. Next, we compute the cartesian product of $Join(\bar{Q}^*_\eta)$ and the $Join(Q^*_{\eta,z})$ of all the $z \in Z$ – namely $\left(\times_{z \in Z} Join(Q^*_{\eta,z})\right) \times Join(\bar{Q}^*_\eta)$ – using p_η machines. Towards that purpose, define for each $z \in Z$

$$p_{\eta,z} = \Theta\left(1 + \max_{k=1}^{|F^*_z|} \frac{P_k(Q^*_{\eta,z}, C^*_z)}{L^k}\right) \tag{9}$$

where $P_k(Q^*_{\eta,z}, C^*_z)$ is the max $(k, Q^*_{\eta,z})$-product of the clustering C^*_z. Similarly, define

$$\bar{p}_\eta = \Theta\left(1 + \max_{k=1}^{|\bar{F}^*|} \frac{P_k(\bar{Q}^*_\eta, \bar{C}^*)}{L^k}\right) \tag{10}$$

where $P_k(\bar{Q}^*_\eta, \bar{C}^*)$ is the max $(k, \bar{Q}^*_\eta)$-product of the clustering $\bar{C}^*$. We will prove later that each $Q^*_{\eta,z}$ can be answered with load $O(L)$ using $p_{\eta,z}$ machines, and $\bar{Q}^*_\eta$ can be answered with load $O(L)$ using $\bar{p}_\eta$ machines. Therefore, applying the cartesian product algorithm given in Lemma 6 of [12] (see also Lemma 4 of [13]), we can compute $\left(\times_{z\in Z} \mathit{Join}(Q^*_{\eta,z})\right) \times \mathit{Join}(\bar{Q}^*_\eta)$ with load $O(L)$ using $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z}$ machines. As proved later, we can adjust the constants in (9) and (10) to make sure $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z} \leq p_\eta$, where p_η is given in (8).

Step 4. We combine the cartesian product $\left(\times_{z\in Z} \mathit{Join}(Q^*_{\eta,z})\right) \times \mathit{Join}(\bar{Q}^*_\eta)$ with the tuples broadcast in Step 2 to derive $\mathit{Join}(Q_\eta)$ with no more communication. Specifically, for each tuple $\boldsymbol{u}$ in the cartesian product, the machine where $\boldsymbol{u}$ resides outputs $\{\boldsymbol{u}\} \bowtie \left(\bowtie_{e\in \text{sigpath}(f^\circ,T)} R(e,\eta)\right)$. It is rudimentary to verify that all the tuples of $\mathit{Join}(Q_\eta)$ will be produced this way.

4 Analysis of the Algorithm

This section will establish:

▶ **Theorem 22.** *Consider any join query Q defined in Section 1.1 whose hypergraph is G. The algorithm of Section 3 answers Q with load $O(L)$, where L (given in (6)) is the Q-induced load of the clustering obtained from a canonical edge cover of G.*

We will prove the theorem by induction on the number of participating attributes (i.e., $|V|$) and the number of participating relations (i.e., $|Q|$). If $|Q| = 1$, the theorem trivially holds. If $|V| = 1$, Q has only one relation (because Q is clean) and the theorem also holds. Next, assuming that the theorem holds on any query with *either* strictly less participating attributes *or* strictly less participating relations than Q, we will prove the theorem's correctness on Q.

Our analysis will answer three questions. First, why do we have enough machines to handle all configurations in parallel? In particular, we must show that $\sum_\eta p_\eta \leq p$, where p_η is given in (8). Second, why does each step in Section 3.2 and 3.3 entail a load of $O(L)$? Third, why do we have $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z} \leq p_\eta$ in Step 3 of Section 3.3? Settling these questions will complete the proof of Theorem 22.

All the notations in this section follow those in Section 3.

4.1 Total Number of Machines for All Configurations

It suffices to prove $\sum_\eta p_\eta = O(p)$ because adjusting the hidden constants then ensures $\sum_\eta p_\eta \leq p$. For every $k \in [|F|]$, we will show

$$\frac{1}{L^k} \sum_\eta P_k(Q_\eta, C) \quad = \quad O(p) \tag{11}$$

which will yield

$$\begin{aligned}
\sum_\eta p_\eta &= \sum_\eta O\left(1 + \max_{k=1}^{|F|} \frac{P_k(Q_\eta, C)}{L^k}\right) \\
&= \sum_\eta O\left(1 + \sum_{k=1}^{|F|} \frac{P_k(Q_\eta, C)}{L^k}\right) = O(p) + \sum_{k=1}^{|F|} O\left(\sum_\eta \frac{P_k(Q_\eta, C)}{L^k}\right) \\
&= O(p)
\end{aligned}$$

where the second equality used $|F| = O(1)$ and the third equality used $\sum_\eta 1 = O(p)$.[8]

[8] $\sum_\eta 1$ is the number of configurations which is $O(p)$ as shown in (7).

Henceforth, fix the value of k. For any η, the hypergraph of Q_η is always G (i.e., the hypergraph of Q). Consider an arbitrary k-group K of the clustering C (given in Equation 5). The Q_η-product of K is $\prod_{e\in K}|R(e,\eta)|$.[9] For any K, we will prove

$$\frac{1}{L^k}\sum_\eta\prod_{e\in K}|R(e,\eta)| \quad = \quad O(p). \tag{12}$$

As C has $O(1)$ k-groups K, the above yields

$$\begin{aligned}\sum_\eta\frac{P_k(Q_\eta,C)}{L^k} &= \sum_\eta\frac{1}{L^k}\max_K\prod_{e\in K}|R(e,\eta)|\\ &= O\left(\sum_\eta\frac{1}{L^k}\sum_K\prod_{e\in K}|R(e,\eta)|\right)=O\left(\sum_K\frac{1}{L^k}\sum_\eta\prod_{e\in K}|R(e,\eta)|\right)\\ &= \sum_K O(p)=O(p)\end{aligned}$$

as claimed in (11).

Let us first consider the case where $K\cap\text{sigpath}(f^\circ,T)\neq\emptyset$, namely, K has a hyperedge e_0 picked from the cluster sigpath(f°,T). We have:

$$\sum_\eta\prod_{e\in K}|R(e,\eta)| \quad = \quad \sum_\eta\left(|R(e_0,\eta)|\cdot\prod_{e\in K\setminus\{e_0\}}|R(e,\eta)|\right) \tag{13}$$

For each $e\in K\setminus\{e_0\}$, obviously $|R(e,\eta)|\leq|R(e)|$. Regarding e_0, because A° must be an attribute of e_0, the relations $R(e_0,\eta)$ of all the configurations η form a *partition* of $R(e_0)$.[10] Hence:

$$\begin{aligned}(13) &\leq \left(\prod_{e\in K\setminus\{e_0\}}|R(e)|\right)\left(\sum_\eta|R(e_0,\eta)|\right)=\left(\prod_{e\in K\setminus\{e_0\}}|R(e)|\right)\cdot|R(e_0)|=\prod_{e\in K}|R(e)|\\ &\leq \max\ (k,Q)\text{-product of } C.\end{aligned}$$

Therefore, the left hand side of (12) is bounded by $\frac{\max\ (k,Q)\text{-product of } C}{L^k}$, which is at most p (by definition of L).

Next, we consider $K\cap\text{sigpath}(f^\circ,T)=\emptyset$. In this case, we must have $k=|K|\leq|F|-1$, because the hyperedges in K need to come from distinct clusters of C, and C has $|F|$ clusters (one of them is sigpath(f°,T), which now must be excluded). Applying the trivial fact $|R(e,\eta)|\leq|R(e)|$ (for any e) and the fact that $\sum_\eta 1$ is bounded by (7), we have

$$\begin{aligned}\frac{1}{L^k}\sum_\eta\prod_{e\in K}|R(e,\eta)| &\leq \frac{1}{L^k}\sum_\eta\prod_{e\in K}|R(e)|=O\left(\frac{1}{L^k}\prod_{e\in K}|R(e)|\cdot\max_{e\in\text{sigpath}(f^\circ,T)}\frac{|R(e)|}{L}\right)\\ &= O\left(\frac{\max\ (k+1,Q)\text{-product of } C}{L^{k+1}}\right)\end{aligned}$$

which is at most p. This completes the proof of $\sum_\eta p_\eta=O(p)$.

[9] For the definition of "a k-group's Q-product", review Section 1.3.
[10] The $R(e_0,\eta)$ of all the η are mutually disjoint and their union equals $R(e_0)$.

4.2 Heavy Q_η

This subsection will prove that the algorithm in Section 3.2 has load $O(L)$. Step 2 and 5 demand no communication. The loads of Step 1 and 3 can all be bounded[11] by $O(\frac{1}{p_\eta}\sum_{e\in E}|R(e,\eta)|) = O(\frac{1}{p_\eta}\max_{e\in E}|R(e,\eta)|) = O(P_1(Q_\eta, C)/p_\eta) = O(L)$.

To analyze Step 4, let T^* be the hyperedge tree of G^* (produced by cleansing) and F^* be the CEC of G^*. By definition, the Q^*_η-induced load of the clustering $C^* = \{\text{sigpath}(f^*, T^*) \mid f^* \in F^*\}$ is

$$L^*_\eta = \max_{k=1}^{|F^*|}\left(\frac{P_k(Q^*_\eta, C^*)}{p_\eta}\right)^{1/k} \tag{14}$$

where $P_k(Q^*_\eta, C^*)$ is the max (k, Q^*_η)-product of C^*. By our inductive assumption (that Theorem 22 holds on Q^*_η), Step 4 incurs load $O(L^*_\eta)$. We will prove $P_k(Q^*_\eta, C^*) \leq P_k(Q_\eta, C)$ for every k which, together with (8) and (14), will tell us $L^*_\eta = O(L)$.

Before proceeding, the reader should recall that, for any hyperedge e^* of G^*, $map^{-1}(e^*)$ gives a hyperedge in G. We must have $|R^*(e^*, \eta)| \leq |R(map^{-1}(e^*), \eta)|$. To see why, note that this is true when $|R^*(e^*, \eta)|$ is created in Step 2, whereas $R^*(e^*, \eta)$ can only shrink in Steps 3-5.

To prove $P_k(Q^*_\eta, C^*) \leq P_k(Q_\eta, C)$, consider any k-group K^* of C^*. By Lemma 15, $K = \{map^{-1}(e^*) \mid e^* \in K^*\}$ must be a k-group of C. Since $|R^*(e^*, \eta)| \leq |R(map^{-1}(e^*), \eta)|$ for any $e^* \in K^*$, we have $\prod_{e^*\in K^*}|R^*(e^*, \eta)| \leq \prod_{e\in K}|R(e, \eta)| \leq P_k(Q_\eta, C)$. Therefore:

$$P_k(Q^*_\eta, C^*) = \max_{K^*}\prod_{e^*\in K^*}|R^*(e^*, \eta)| \leq P_k(Q_\eta, C).$$

4.3 Light Q_η

This subsection will concentrate on the algorithm of Section 3.3.

Load. Step 1 incurs load $O(L)$ (same analysis as in Section 3.2). Step 2 also requires a load of $O(L)$ because every broadcast relation has a size of at most L. Step 4 needs no communication.

To analyze Step 3, let us first consider $\bar{Q}^*_\eta$. The $\bar{Q}^*_\eta$-induced load of the clustering $\bar{C}^*$ is

$$\bar{L}^*_\eta = \max_{k=1}^{|\bar{C}^*|}\left(\frac{P_k(\bar{Q}^*_\eta, \bar{C}^*)}{\bar{p}_\eta}\right)^{1/k}$$

where $P_k(\bar{Q}^*_\eta, \bar{C}^*)$ as the max $(k, \bar{Q}^*_\eta)$-product of $\bar{C}^*$. By our inductive assumption (that Theorem 22 holds on $\bar{Q}^*_\eta$), answering $\bar{Q}^*_\eta$ with $\bar{p}_\eta$ machines requires load $O(\bar{L}^*_\eta)$, which is $O(L)$ given the $\bar{p}_\eta$ in (10). A similar argument shows that answering each $Q^*_{\eta,z}$ with $p_{\eta,z}$ machines – with $p_{\eta,z}$ given in (9) – incurs a load of $O(L)$. Thus, the cartesian product at Step 3 can be computed with load $O(L)$.

Number of machines in Step 3. Next, we will prove that $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z} \leq p_\eta$ always holds in Step 3. It suffices to show $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z} = O(p_\eta)$ which, as we will see, relies on Lemma 20 and the fact that $|R(e, \eta)| \leq L$ for every node $e \in \text{sigpath}(f^\circ, T)$.

[11] Step 3 performs $O(1)$ semi joins, each of which can be performed by sorting. For sorting in the MPC model, see Section 2.2.1 of [10]. The stated bound for Step 1 and 3 requires the assumption $p \leq m^{1-\epsilon}$.

Consider an arbitrary $z \in Z$. The root of T_z^* – denoted as e_{root} – must belong to sigpath(f°, T). Recall that a k-group K of C_z^* takes a hyperedge from a distinct cluster in C_z^*. Call K a *non-root k-group* if $e_{root} \notin K$, or a *root k-group*, otherwise. Define

$$\begin{aligned} P_k(Q_{\eta,z}^*, C_z^*) &= \max\ (k, Q_{\eta,z}^*)\text{-product of } C_z^* \\ P_k^{non}(Q_{\eta,z}^*, C_z^*) &= \max\ (k, Q_{\eta,z}^*)\text{-product of all the } \textit{non-root } k\text{-groups of } C_z^*. \end{aligned}$$

As a special case, define $P_0^{non}(Q_{\eta,z}^*, C_z^*) = 1$. For any k, we observe

$$P_k(Q_{\eta,z}^*, C_z^*) \leq \max\{P_k^{non}(Q_{\eta,z}^*, C_z^*), L \cdot P_{k-1}^{non}(Q_{\eta,z}^*, C_z^*)\}. \tag{15}$$

To prove the inequality, fix K to the k-group with the largest $Q_{\eta,z}^*$-product $(= P_k(Q_{\eta,z}^*, C_z^*))$. If K is a non-root k-group, (15) obviously holds. Consider, instead, that K is a root k-group. Since $e_{root} \in$ sigpath(f°, T), we know $|R(e_{root}, \eta)| \leq L$ and hence $\prod_{e \in K} |R(e, \eta)| \leq L \cdot \prod_{e \in K \setminus \{e_{root}\}} |R(e, \eta)|$. As $K \setminus \{e_{root}\}$ is a non-root $(k-1)$-group, $P_k(Q_{\eta,z}^*, C_z^*) \leq L \cdot P_{k-1}^{non}(Q_{\eta,z}^*, C_z^*)$ holds.

Equipped with (15), we can now derive from (9):

$$\begin{aligned} p_{\eta,z} &= O\left(1 + \max_{k=1}^{|F_z^*|} \frac{\max\{P_k^{non}(Q_{\eta,z}^*, C_z^*), L \cdot P_{k-1}^{non}(Q_{\eta,z}^*, C_z^*)\}}{L^k}\right) \\ &= O\left(1 + \max_{k=1}^{|F_z^*|-1} \frac{P_k^{non}(Q_{\eta,z}^*, C_z^*)}{L^k}\right) \end{aligned} \tag{16}$$

where the second equality used the fact that, when $k = |F_z^*|$, a k-group must be a root k-group.

We are now ready to prove $\bar{p}_\eta \cdot \prod_{z \in Z} p_{\eta,z} = O(p_\eta)$. For each $z \in Z$, define integer k_z and a set K_z of hyperedges as follows:

- If (16) $= \Theta(P_k^{non}(Q_{\eta,z}^*, C_z^*)/L^k)$ for some $k \in [1, |F_z^*| - 1]$, set $k_z = k$ and K_z to the non-root k-group whose $Q_{\eta,z}^*$-product equals $P_k^{non}(Q_{\eta,z}^*, C_z^*)$.
- Otherwise (we must have $p_{\eta,z} = \Theta(1)$), set $k_z = 0$ and $K_z = \emptyset$; furthermore, define the $Q_{\eta,z}^*$-product of K_z to be 1.

Similarly, regarding $\bar{p}_\eta$ in (10), define integer $\bar{k}$ and a set $\bar{K}$ of hyperedges as follows:

- If (10) $= \Theta(P_k(\bar{Q}_\eta^*, \bar{C}^*)/L^k)$ for some $k \in [1, |\bar{F}^*|]$, set $\bar{k} = k$ and $\bar{K}$ to the k-group of the clustering $\bar{C}^*$ whose $\bar{Q}_\eta^*$-product equals $P_k(\bar{Q}_\eta^*, \bar{C}^*)$.
- Otherwise, set $\bar{k} = 0$ and $\bar{K} = \emptyset$; furthermore, define the $\bar{Q}_\eta^*$-product of $\bar{K}$ to be 1.

Define $K_{super} = \bar{K} \cup \left(\bigcup_{z \in Z} K_z\right)$. If $K_{super} = \emptyset$, then $p_{\eta,z} = \Theta(1)$ for all $z \in Z$ and $\bar{p}_\eta = \Theta(1)$, which leads to

$$\bar{p}_\eta \cdot \prod_{z \in Z} p_{\eta,z} = O(1) = O(p_\eta).$$

If $K_{super} \neq \emptyset$, K_{super} is a super-$|K_{super}|$-group[12]. By Lemma 20, K_{super} is a $|K_{super}|$-group of T. We thus have:

$$\begin{aligned} \bar{p}_\eta \cdot \prod_{z \in Z} p_{\eta,z} &= \frac{\bar{Q}_\eta^*\text{-product of } \bar{K}}{L^{\bar{k}}} \prod_{z \in Z} \frac{Q_{\eta,z}^*\text{-product of } K_z}{L^{k_z}} \\ &= \frac{\prod_{e \in K_{super}} |R(e)|}{L^{|K_{super}|}} \leq \frac{\max\ (|K_{super}|, Q_\eta)\text{-product of } C}{L^{|K_{super}|}} = O(p_\eta). \end{aligned}$$

This completes the whole proof of Theorem 22.

[12] For the definition of super-k-group, review Section 2.2.2.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Foto N. Afrati, Manas R. Joglekar, Christopher Ré, Semih Salihoglu, and Jeffrey D. Ullman. GYM: A multiround distributed join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 4:1–4:18, 2017.

3 Foto N. Afrati and Jeffrey D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(9):1282–1298, 2011.

4 Kaleb Alway, Eric Blais, and Semih Salihoglu. Box covers and domain orderings for beyond worst-case join processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 3:1–3:23, 2021.

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

6 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):40:1–40:58, 2017.

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 Xiao Hu. Cover or pack: New upper and lower bounds for massively parallel joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 181–198, 2021.

9 Xiao Hu and Ke Yi. Instance and output optimal parallel algorithms for acyclic joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 450–463, 2019.

10 Xiao Hu, Ke Yi, and Yufei Tao. Output-optimal massively parallel algorithms for similarity joins. *ACM Transactions on Database Systems (TODS)*, 44(2):6:1–6:36, 2019.

11 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 1259–1274. ACM, 2017.

12 Bas Ketsman and Dan Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 417–428, 2017.

13 Bas Ketsman, Dan Suciu, and Yufei Tao. A near-optimal parallel algorithm for joining binary relations. *CoRR*, abs/2011.14482, 2020.

14 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 213–228, 2015.

15 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-case optimal algorithms for parallel query processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 8:1–8:18, 2016.

16 Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 234–245, 2014.

17 Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 37–48, 2012.

18 Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):16:1–16:40, 2018.

19 Hung Q. Ngo, Christopher Re, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013.

20 Miao Qiao and Yufei Tao. Two-attribute skew free, isolated CP theorem, and massively parallel joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 166–180, 2021.

21 Yufei Tao. A simple parallel algorithm for natural joins on binary relations. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 25:1–25:18, 2020.

22 Yufei Tao. Parallel acyclic joins with canonical edge covers. *CoRR*, abs/2201.03832, 2022.

23 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of Very Large Data Bases (VLDB)*, pages 82–94, 1981.

Splitting Spanner Atoms: A Tool for Acyclic Core Spanners

Dominik D. Freydenberger
Loughborough University, UK

Sam M. Thompson
Loughborough University, UK

Abstract

This paper investigates regex CQs with string equalities (SERCQs), a subclass of core spanners. As shown by Freydenberger, Kimelfeld, and Peterfreund (PODS 2018), these queries are intractable, even if restricted to acyclic queries. This previous result defines acyclicity by treating regex formulas as atoms. In contrast to this, we propose an alternative definition by converting SERCQs into FC-CQs – conjunctive queries in FC, a logic that is based on word equations. We introduce a way to decompose word equations of unbounded arity into a conjunction of binary word equations. If the result of the decomposition is acyclic, then evaluation and enumeration of results become tractable. The main result of this work is an algorithm that decides in polynomial time whether an FC-CQ can be decomposed into an acyclic FC-CQ. We also give an efficient conversion from synchronized SERCQs to FC-CQs with regular constraints. As a consequence, tractability results for acyclic relational CQs directly translate to a large class of SERCQs.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Document spanners, information extraction, conjunctive queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.10

Related Version *Full Version*: https://arxiv.org/abs/2104.04758

Funding *Dominik D. Freydenberger*: Supported by EPSRC grant EP/T033762/1.

Acknowledgements The authors would like to thank Justin Brackemann, and the anonymous reviewers for all their helpful comments and suggestions.

1 Introduction

Document spanners were introduced by Fagin, Kimelfeld, Reiss, and Vansummeren [7] as a formalization of AQL, an information extraction query language used in IBM's SystemT. Informally, they can be described in two steps. First, so-called *extractors* convert an input document, a word over a finite alphabet, into relations of so-called *spans*. We assume the extractors to be *regex formulas* (as described in [7]), which are regular expressions with capture variables. Consider the following example of a regex formula

$$\gamma(x) := \Sigma^* \cdot x\{(\texttt{EBDT}) \vee (\texttt{ICDT})\} \cdot \Sigma^*.$$

Given some input word, $\gamma(x)$ can be used to extract a unary relation of spans such that each span represents a factor of the input word that is either "EBDT" or "ICDT".

The second step is that the extracted relations are combined using a relational algebra. Classes of spanners can be defined by the choice of relational operators. *Regular spanners* allow for union $\cup$, projection π, and natural join $\bowtie$. Depending on how they are represented, regular spanners have been shown to be efficient. For example, if a regular spanner is given as a so-called vset-automaton, results can be enumerated with constant delay after

linear time preprocessing [9, 2]. However, if a regular spanner is given as a join of regex formulas, evaluation is intractable – as shown in [13], evaluation for spanners of the form $P := \pi_\emptyset(\gamma_1 \bowtie \gamma_2 \cdots \bowtie \gamma_n)$ is NP-complete, even if P is acyclic.

Core spanners extend regular spanners by allowing equality selection $\zeta^=$, which checks whether two (potentially different) spans represent the same factor of the input document. Even when core spanners are restricted to queries of the form $\pi_\emptyset \zeta^=_{x_1,y_1} \cdots \zeta^=_{x_m,y_m} \gamma$ for a single regex formula γ, the evaluation problem is NP-complete [11]. Therefore, both joins and equalities introduce computational hardness.

Regex CQs can be understood as the spanner version of relational CQs, which are a central topic in database theory. In each case, a conjunctive query is a projection over a join of atoms. Apart from the setting, the key difference is that while the tables for relational CQs are usually part of the input, the tables for regex CQs are defined implicitly through the regex formulas. Hence, while one could extract these tables and then perform a standard CQ over the extractions, the number of tuples in the materialized relations may be exponential. As a consequence, tractable restrictions on relational queries (such as *acyclic* CQ*s*) do not lead to tractable fragments of regex CQs [13].

So-called SERCQs extend regex CQs by also allowing string equality, thus allowing us to examine both previously discussed sources of intractability. Consider the following SERCQ

$$P := \pi_{x,y}\, \zeta^=_{x,x'} \left(\gamma_{\mathsf{sen}}(z) \bowtie \gamma_{\mathsf{prod}}(x) \bowtie \gamma_{\mathsf{pos}}(y) \bowtie \gamma_{\mathsf{factors}}(x, x', z) \bowtie \gamma_{\mathsf{factor}}(y, z)\right),$$

where we assume γ_{sen} extracts sentences, γ_{prod} extracts product names, γ_{pos} extracts positive sentiments (such as "enjoyed"), and $\gamma_{\mathsf{factors}}(x, x', z)$ and $\gamma_{\mathsf{factor}}(y, z)$ ensure that x and x' are successive (but not necessarily consecutive) factors of z, and y is a factor of z respectively. Therefore, P extracts spans representing products that are mentioned twice within a sentence, along with a positive sentiment that appears in the same sentence.

Syntactic restrictions on conjunctive queries have been incredibly fruitful for finding tractable fragments. A well known result of Yannakakis [22] is that for *acyclic* conjunctive queries, evaluation can be solved in polynomial time. Further research on the complexity of acyclic conjunctive queries [15] and the enumeration of results for acyclic conjunctive queries [3] has shown the efficacy of this restriction. On the other hand, for document spanners, such syntactic restrictions are yet to unlock tractable fragments.

To address this gap, we consider a different approach and represent SERCQs as a conjunctive query fragment of the logic FC[REG], introduced by Freydenberger and Peterfreund [14]. This logic is based on word equations, regular constraints, and first-order logic connectives. Consider the following FC[REG] conjunctive query

$$\varphi := \mathsf{Ans}(x, y) \leftarrow (z \doteq z_2 \cdot x \cdot z_3 \cdot x \cdot z_4) \wedge (z \doteq z_5 \cdot y \cdot z_6) \wedge (z \mathrel{\dot\in} \gamma_{\mathsf{sen}}) \wedge (x \mathrel{\dot\in} \gamma_{\mathsf{prod}}) \wedge (y \mathrel{\dot\in} \gamma_{\mathsf{pos}}).$$

If γ_{sen} is a regular expression that accepts sentences, γ_{prod} accepts a product name, and γ_{pos} accepts a positive sentiment, then φ is "equivalent" to the previously given SERCQ. They are not equivalent in a strict sense – a key difference being that SERCQs reason over spans, whereas FC[REG]-CQs reason over factors of the input words. Reasoning over words does bring some advantages: For example, φ simply uses relations of words (for example, γ_{prod}) encoded as a regular expression, and if we wanted to do something analogous for regex-formulas, we would first have to extract the corresponding relation of spans.

When dealing with word equations, we run into an issue that we already encountered for regex formulas: Their relations may contain an exponential number of tuples. This is due to the unbounded arity of word equations. However, an FC atom can be considered shorthand for a concatenation term. For example, the word equation $y \doteq x_1 x_2 x_3 x_4$ can be represented

as $y \doteq f(f(x_1, x_2), f(x_3, x_4))$ where f denotes binary concatenation. This then lends itself to the "decomposition" of the word equation into a CQ consisting of smaller word equations. We can express the above word equation as $(y \doteq z_1 \cdot z_2) \wedge (z_1 \doteq x_1 \cdot x_2) \wedge (z_2 \doteq x_3 \cdot x_4)$. For such a *decomposition*, the relations defined by each word equation can be stored in linear space and we can enumerate them with constant delay. Thus, if the resulting query is acyclic, then the tractability properties of acyclic conjunctive queries directly translate to the FC-CQ.

Contributions of this paper. The goal of this work is to bridge the gap between acyclic relational CQs and information extraction. To this end, we define FC[REG]-CQs, a conjunctive query fragment of FC[REG], and show show that any so-called synchronized SERCQ can be converted into an equivalent FC[REG]-CQ in polynomial time (Lemma 3.6).

We define the decomposition of an FC-CQ into a 2FC-CQ, where 2FC-CQ denotes the set of FC-CQs where the right-hand side of each word equation is of at most length two. Our first main result is a polynomial-time algorithm that decides whether a *pattern*[1] can be decomposed into an acyclic 2FC-CQ (Theorem 4.12).

Building on this, we give a polynomial-time algorithm that decomposes an FC-CQ into an acyclic 2FC-CQ, or determines that this is not possible (Theorem 5.14). As soon as we have an acyclic 2FC-CQ, the upper bound results for model checking and enumeration of results follow from previous work on relational acyclic CQs [15, 3].

We mainly focus on FC-CQs (i. e., no regular constraints) due to the fact that we can add regular constraints for "free". This is because regular constraints are unary predicates, and therefore can be easily incorporated into a join tree. Thus, our work defines a class of FC[REG]-CQs for which model checking can be solved in polynomial time, and results can be enumerated with polynomial-delay (both in terms of combined complexity).

Our approach offers a new research direction for tractable document spanners. Most of the current literature approaches regular spanners by "compiling" the spanner representation (regex formulas that are combined with projection, union, and joins) into a single automaton, where the use of joins can lead to a number of states that is exponential in the size of the original representation. Instead, we look at decomposing FC conjunctive queries into small and tractable components. This allows us to use the wealth of research on relational algebra, while also allowing for the use of the string equality selection operator.

Related Work. Regarding data complexity, Florenzano, Riveros, Vgarte, Vansummeren, and Vrgoc [9] gave a constant-delay algorithm for enumerating the results of deterministic vset-automata, after linear time preprocessing. Amarilli, Bourhis, Mengal, and Niewerth [2] extended this result to non-deterministic vset-automata. Regarding combined complexity, Freydenberger, Kimelfeld, and Peterfreund [13] introduced regex CQs and proved that their evaluation is NP-complete (even for acyclic queries), and that fixing the number of atoms and the number of string equalities in SERCQs allows for polynomial-delay enumeration of results. Freydenberger, Peterfreund, Kimelfeld, and Kröll [12] showed that non-emptiness for a join of two sequential regex formulas is NP-hard, under schemaless semantics, even for a single character document. Connections between the theory of concatenation and spanners have been considered in [11, 10, 14], which give many of the lower bound complexity results for core spanners. Schmid and Schweikardt [21] examined a subclass of core spanners called refl-spanners, which incorporate string equality directly into a regular spanner. Peterfreund [19] considered extraction grammars, and gave an algorithm for unambiguous extraction grammars that enumerates results with constant-delay after quintic preprocessing.

[1] For the purposes of this introduction, a pattern can be considered a single FC atom.

2 Preliminaries

Let $\emptyset$ denote the *empty set*, and for $n \geq 1$ let $[n] := \{1, 2, \ldots, n\}$. Given a set S, we use $|S|$ for the *cardinality* of S. If S is a subset of T then we write $S \subseteq T$ and if $S \neq T$ also holds, then $S \subset T$. We write $\mathcal{P}(S)$ for the powerset of S. The difference of two sets S and T is denoted as $S \setminus T$. If $\vec{x}$ is a tuple, we write $x \in \vec{x}$ to indicate that x is a component of $\vec{x}$. Let A be an alphabet. We use $|w|$ to denote the length of some word $w \in A^*$ and ε to denote the *empty word*. The number of occurrences of $a \in A$ within w is $|w|_a$. We write $u \cdot v$ or just uv for the concatenation of words $u, v \in A^*$. If $u = p \cdot v \cdot s$ for $p, s \in A^*$ then v is a *factor* of u, denoted $v \sqsubseteq u$. If $u \neq v$ also holds, then $v \sqsubset u$. Let Σ be an alphabet of *terminal symbols* and let Ξ be an infinite alphabet of *variables*. We assume that $\Sigma \cap \Xi = \emptyset$ and $|\Sigma| \geq 2$.

If $T := (V, E)$ is a tree, then a path between $x_1 \in V$ and $x_n \in V$ is the shortest sequence of edges from x_1 to x_n. If $(\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_{n-1}, x_n\})$ is a path, then we say a node y *lies* on this path if $y = x_j$ for some $j \in [n]$. We call the number of edges on a path from x_1 to x_n the *distance* between x_1 and x_n.

Document Spanners. Given $w := w_1 \cdot w_2 \cdots w_n$ where $w_i \in \Sigma$ for all $i \in [n]$, a so-called *span* of w is an interval $[i, j\rangle$ where $1 \leq i \leq j \leq n + 1$. A span $[i, j\rangle$ defines a factor $w_{[i,j\rangle} := w_i \cdot w_{i+1} \cdots w_{j-1}$ of w. Let $V \subset \Xi$, where V is finite, and let $w \in \Sigma^*$. A (V, w)-*tuple* is a function μ that maps each $x \in V$ to a span $\mu(x)$ of w. A *spanner* P, with variables V, is a function that maps every $w \in \Sigma^*$ to a set $P(w)$ of (V, w)-tuples. By $\mathsf{Vars}(P)$, we denote the set of variables of P.

Like [7], we use *regex formulas* as the primary extractors. Regex formulas are an extension of regular expressions with so-called *capture variables*. More formally: $\emptyset$, ε, and $\mathtt{a}$ where $\mathtt{a} \in \Sigma$ are all regex formulas, and if γ_1 and γ_2 are regex formulas then so are $(\gamma_1 \cdot \gamma_2)$, $(\gamma_1 \vee \gamma_2)$, $(\gamma_1)^*$, and $x\{\gamma_1\}$ where $x \in \Xi$. We use Σ as a shorthand for $\bigvee_{a \in \Sigma} a$. We can omit the parentheses when the meaning is clear. A variable binding $x\{\gamma\}$ matches the same words as γ and assigns the corresponding span of the input word to x. A regex formula is *functional* if on every match, each variable is assigned exactly one span. We denote the set of functional regex formulas by RGX. For $\gamma \in \mathsf{RGX}$, we use $[\![\gamma]\!]$ to define the corresponding spanner as follows. Every match of γ on w defines μ, a $(\mathsf{Vars}(\gamma), w)$-tuple, where for each $x \in \mathsf{Vars}(\gamma)$, we have that $\mu(x)$ is the span assigned to x. We use $[\![\gamma]\!](w)$ to denote the set of all such $(\mathsf{Vars}(\gamma), w)$-tuples. See [7] for more details.

We now define *synchronized* RGX*-formulas* (this follows the definition by Freydenberger, Kimelfeld, Kröll, and Peterfreund in [12]). An expression $\gamma \in \mathsf{RGX}$ is *synchronized* if for all sub-expressions of the form $(\gamma_1 \vee \gamma_2)$, no variable bindings occur in γ_1 or γ_2. We denote the class of synchronized RGX-formulas by $\mathsf{RGX}_{\mathsf{sync}}$.

The motivation for synchronized RGX-formulas is that non-synchronized formulas allow for "hidden" disjunctions within the atoms. This goes (arguably) against the spirit of CQs and (as shown in [12]) leads to "un-CQ-like" behavior.

▶ **Example 2.1.** Consider the regex formula $\gamma := \Sigma^* \cdot x\{\mathtt{a} \vee (\mathtt{b})^*\} \cdot y\{\Sigma^*\} \cdot \Sigma^*$. We have that $[\![\gamma]\!](w)$ contains those μ such that $\mu(x)$ is a factor of w which is either an $\mathtt{a}$ or a sequence of $\mathtt{b}$ symbols, and the span $\mu(y)$ occurs directly after $\mu(x)$. Since γ is functional, and for every sub-expression of the form $(\gamma_1 \vee \gamma_2)$, we have that $\mathsf{Vars}(\gamma_1) = \mathsf{Vars}(\gamma_2) = \emptyset$, it follows that γ is a *synchronized regex formula*.

Essentially, a synchronized regex formula is functional if no variable is redeclared, and no variable is used inside of a Kleene star.

This is extended into a *relational algebra* comprised of $\cup$ (union), π (projection), $\bowtie$ (natural join), and $\zeta^{=}$ (string equality). Let $w \in \Sigma^*$, and let P_1 and P_2 be spanners. We say P_1 and P_2 are *compatible* if $\mathsf{Vars}(P_1) = \mathsf{Vars}(P_2)$. If two spanners P_1 an P_2 are compatible, then $(P_1 \cup P_2)(w) := P_1(w) \cup P_2(w)$. For $Y \subseteq \mathsf{Vars}(P_1)$, the *projection* $\pi_Y P_1(w)$ is defined as the restriction of all $\mu \in P_1(w)$ to the set of variables Y, and hence $\mathsf{Vars}(\pi_Y P_1) := Y$.

The *natural join*, $P_1 \bowtie P_2$, is obtained by defining $\mathsf{Vars}(P_1 \bowtie P_2) := \mathsf{Vars}(P_1) \cup \mathsf{Vars}(P_2)$, and $(P_1 \bowtie P_2)(w)$ as the set of all $(\mathsf{Vars}(P_1) \cup \mathsf{Vars}(P_2), w)$-tuples for which there exists $\mu_1 \in P_1(w)$ and $\mu_2 \in P_2(w)$ such that $\mu_1(x) = \mu_2(x)$ for all $x \in \mathsf{Vars}(P_1) \cap \mathsf{Vars}(P_2)$. The *string equality operator* $\zeta^{=}_{x_1,x_2} P_1$ is defined by $\zeta^{=}_{x_1,x_2} P_1(w) := \{\mu \in P_1(w) \mid w_{\mu(x_1)} = w_{\mu(x_2)}\}$, where $\mathsf{Vars}\left(\zeta^{=}_{x_1,x_2} P_1\right) := \mathsf{Vars}(P_1)$.

Given a class of regex-formulas C and a spanner algebra O, we use C^{O} to denote the set of spanner representations which can be constructed by repeated combinations of operators from O with a regex-formula from C. We write $[\![C^{\mathsf{O}}]\!]$ to denote the closure of $[\![C]\!]$ under O.

The class of *core spanners* (introduced by Fagin, Kimelfeld, Reiss, and Vansummeren [7]) is defined as $[\![\mathsf{RGX}^{\mathsf{core}}]\!]$ where $\mathsf{core} := \{\pi, \zeta^{=}, \cup, \bowtie\}$. The class of *regex* CQ*s with string equality* (SERCQs) is defined as expressions of the form:

$$P := \pi_Y \left(\zeta^{=}_{x_1,y_1} \cdots \zeta^{=}_{x_l,y_l} (\gamma_1 \bowtie \cdots \bowtie \gamma_k)\right),$$

where $\gamma_i \in \mathsf{RGX}$ for all $i \in [k]$. We call an SERCQ a *synchronized* SERCQ if every regex formula is a synchronized RGX-formula.

▶ **Example 2.2.** Consider $P := \zeta^{=}_{x_1,x_2}(\gamma_1 \bowtie \gamma_2)$ where $\gamma_1 := \Sigma^* \cdot x_1\{\Sigma^+\} \cdot \mathtt{a} \cdot \Sigma^*$ and $\gamma_2 := \Sigma^* \cdot x_2\{\Sigma^+\} \cdot \mathtt{b} \cdot \Sigma^*$. Given $w \in \Sigma^*$, we have that $[\![P]\!](w)$ contains those μ such that the factor $w_{\mu(x_1)}$ is non-empty, and is immediately followed by the symbol $\mathtt{a}$, the factor $w_{\mu(x_2)}$ is immediately followed by the symbol $\mathtt{b}$, and $w_{\mu(x_1)} = w_{\mu(x_2)}$. Since both γ_1 and γ_2 are synchronized, P is a synchronized SERCQ.

Computational Model and Complexity Measures. We use the *random access machine* model with uniform cost measures, where the size of each machine word is logarithmic in the size of the input. We represent factors of a word $w \in \Sigma^*$ as spans of w. This allows us to check whether $u = v$ for $u, v \sqsubseteq w$ in constant time after preprocessing that takes linear time and space [16, 5] (see Proposition 4.1 for more details). The complexity results we state are in terms of *combined complexity*. That is, both the query and the word are considered part of the input. When considering the enumeration of results for a query executed on a word, we say that we can enumerate results with *polynomial-delay* if there exists an algorithm which returns the first result in polynomial time, the time between two consecutive results is polynomial, and the time between the last result and terminating is polynomial.

3 Conjunctive Queries for FC

This section introduces FC[REG]-CQs, a conjunctive query fragment of FC with regular constraints. We give some complexity results regarding SERCQs and show an efficient conversion from synchronized SERCQs to FC[REG]-CQs.

A pattern is a word $\alpha \in (\Sigma \cup \Xi)^*$, and a *word equation* is a pair $\eta := (\alpha_L, \alpha_R)$ where $\alpha_L, \alpha_R \in (\Sigma \cup \Xi)^*$ are patterns known as the *left* and *right* side respectively. We usually write such η as $(\alpha_L \doteq \alpha_R)$. The length of a word equation, denoted $|(\alpha_L \doteq \alpha_R)|$, is $|\alpha_L| + |\alpha_R|$. A *pattern substitution* is a morphism $\sigma\colon (\Sigma \cup \Xi)^* \to \Sigma^*$ such that $\sigma(\mathtt{a}) = \mathtt{a}$ holds for all $\mathtt{a} \in \Sigma$. Since σ is a morphism, we have $\sigma(\alpha_1 \cdot \alpha_2) = \sigma(\alpha_1) \cdot \sigma(\alpha_2)$ for all $\alpha_1, \alpha_2 \in (\Sigma \cup \Xi)^*$.

A pattern substitution σ is a *solution* to a word equation $(\alpha_L \doteq \alpha_R)$ if and only if $\sigma(\alpha_L) = \sigma(\alpha_R)$. When applying a pattern substitution σ to a pattern α, we assume that its domain $\mathsf{dom}(\sigma)$ satisfies $\mathsf{var}(\alpha) \subseteq \mathsf{dom}(\sigma)$. Freydenberger and Peterfreund [14] introduced FC as a first-order logic that is based on word equations. In the present paper, we do not consider the full logic FC. Instead, we introduce its conjunctive queries.

▶ **Definition 3.1.** *An* FC-CQ *is an* FC*-formula of the form* $\varphi(\vec{x}) := \exists \vec{y}\colon \bigwedge_{i=1}^{n} \eta_i$*, where* $\eta_i := (x_i \doteq \alpha_i)$*,* $x_i \in \Xi$*, and* $\alpha_i \in (\Sigma \cup \Xi)^*$ *for all* $i \in [n]$*. We use the shorthand* $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ *where* $\vec{x}$ *is the tuple of free variables. We call* $\mathsf{Ans}(\vec{x})$ *the* head *of* φ*, and* $\bigwedge_{i=1}^{n} \eta_i$ *the* body *of* φ*.*

We write $\varphi(\vec{x})$ to denote that $\vec{x}$ is the set of free variables of φ. The set of all variables used in φ is denoted by $\mathsf{var}(\varphi)$. We distinguish a variable $\mathfrak{u} \in \Xi$, called the *universe variable*, that shall represent the input document w. The universe variable is not considered a free variable, and we adopt the convention that $\mathfrak{u} \notin \mathsf{var}(\varphi)$ for all φ (even if $\mathfrak{u}$ occurs in φ). Next, we define the semantics for FC-CQs.

▶ **Definition 3.2.** *For* $\varphi \in$ FC-CQ *and a pattern substitution* σ *with* $\mathsf{var}(\varphi) \cup \{\mathfrak{u}\} \subseteq \mathsf{dom}(\sigma)$*, we define* $\sigma \models \varphi$ *as follows:* $\sigma \models (\alpha_l \doteq \alpha_R)$ *if* $\sigma(\eta_L) = \sigma(\eta_R)$ *and* $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$ *for all* $x \in \mathsf{var}(\alpha_L \doteq \alpha_R)$*. For* $\sigma \models \exists x\colon \varphi$ *we have that* $\sigma_{x \mapsto u} \models \varphi$ *holds for some* $u \sqsubseteq \sigma(\mathfrak{u})$*, where* $\sigma_{x \mapsto u}$ *is defined as* $\sigma_{x \mapsto u}(x) := u$ *and* $\sigma_{x \mapsto u}(y) := \sigma(y)$ *for all* $y \in (\Sigma \cup \Xi)$ *where* $y \neq x$*. We use the canonical definition for conjunction.*

Hence, for all $\sigma \models \varphi(\vec{x})$, the universe for variables in $\mathsf{var}(\varphi)$ is the set of factors of $\sigma(\mathfrak{u})$. If $\varphi(\vec{x}) \in$ FC-CQ and $w \in \Sigma^*$, then $[\![\varphi]\!](w)$ denotes the set of all $\sigma(\vec{x})$ such that $\sigma \models \varphi$ and $\sigma(\mathfrak{u}) = w$. When determining $[\![\varphi]\!](w)$ for a given w, we know that $\mathfrak{u}$ represents w, and hence $\mathfrak{u}$ can be treated as a constant (see [14] for more information on the role of the universe variable). If $\varphi \in$ FC is *Boolean* (that is, it has no free variables), $[\![\varphi]\!](w)$ is either the empty set, or the set containing the empty tuple, which we interpret as False and True, respectively.

In [14], FC was extended to FC[REG] by adding *regular constraints*. This allows for atoms of the form $(x \mathrel{\dot\in} \gamma)$, where γ is a *regular expression*; and $\sigma \models (x \mathrel{\dot\in} \gamma)$ if and only if $\sigma(x) \in \mathcal{L}(\gamma)$ and $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$. We extend FC-CQ to FC[REG]-CQ in the same way.

Complexity. We now define various decision problems for FC-CQ and FC[REG]-CQ: The *non-emptiness problem* is, given $w \in \Sigma^*$ and φ, decide whether $[\![\varphi]\!](w) \neq \emptyset$. The *evaluation problem* is, given σ and φ, decide whether $\sigma \models \varphi$. The *model checking problem* is the special case of non-emptiness and evaluation that only considers Boolean queries, note that for Boolean queries $\mathsf{Dom}(\sigma) = \{\mathfrak{u}\}$. Given $w \in \Sigma^*$ and φ, the *enumeration problem* is outputting all $[\![\varphi]\!](w)$. The *containment problem* is, given φ and ψ, decide whether $[\![\varphi]\!](w) \subseteq [\![\psi]\!](w)$ for all $w \in \Sigma^*$. Previous results on patterns and FC (see [4, 6, 14]) directly imply the following.

▶ **Proposition 3.3.** *For each of* FC-CQ *and* FC[REG]-CQ*, the evaluation problem is* NP*-complete, and the containment problem is undecidable.*

As discussed in [14], FC and FC[REG] can be evaluated analogously to relational first-order logic (FO), by materializing the tables that are defined by the atoms and then proceeding "as usual". Hence, bounding the width of a formula (the maximum number of free variables in a subformula) bounds the size of the intermediate tables, and thereby the complexity of evaluation. As the complexity of evaluating FC and FO are the same (PSPACE-complete in general, NP-complete for the existential-positive fragment), it is no surprise that this correspondence also translates to conjunctive queries. From Section 5 on, we further develop this connection by finding tractable subclasses of FC[REG]-CQ.

As containment for CQs is decidable (although NP-complete), it can be used for query minimization (see Chapter 6 of [1]). But by Proposition 3.3, this does not apply to FC-CQ.

Document Spanners and FC-CQs. Our next goal is to establish a connection between SERCQs and FC[REG]-CQs. However, first we must overcome the fact that FC reasons over strings, whereas spanners reason over intervals of positions. We deal with this by defining the notion of an FC-formula *realizing* a spanner, as described in [11, 10, 14].

▶ **Definition 3.4.** *A pattern substitution* σ expresses *a* (V, w)*-tuple* μ*, if for all* $x \in V$*, we have that* $\mathsf{Dom}(\sigma) = \{x^P, x^C \mid x \in V\}$*, and* $\sigma(x^P) = w_{[1,i\rangle}$ *and* $\sigma(x^C) = w_{[i,j\rangle}$ *for the span* $\mu(x) = [i, j\rangle$*. An* FC[REG]-CQ φ realizes *a spanner* P *if* $\mathsf{free}(\varphi) = \{x^P, x^C \mid x \in \mathsf{Vars}(P)\}$ *and* $\sigma \models \varphi$ *for all* $w \in \Sigma^*$ *where* $\sigma(\mathfrak{u}) = w$*, if and only if* σ *expresses some* $\mu \in P(w)$*.*

Less formally, for each $\mu \in P(w)$, we have that $\mu(x) = [i, j\rangle$ is uniquely represented by the prefix, $\sigma(x^P) = w_{[1,i\rangle}$, and the content, $\sigma(x^C) = w_{[i,j\rangle}$.

▶ **Example 3.5.** Consider the following FC[REG]-CQ.

$$\varphi := \mathsf{Ans}(x_1^P, x_1^C, x_2^P, x_2^C) \leftarrow (\mathfrak{u} \doteq x_1^P \cdot x_1^C \cdot \mathsf{a} \cdot s_1) \wedge (\mathfrak{u} \doteq x_2^P \cdot x_2^C \cdot \mathsf{b} \cdot s_2) \\ \wedge (x_1^C \doteq x_2^C) \wedge (x_1^C \mathrel{\dot\in} \Sigma^+) \wedge (x_2^C \mathrel{\dot\in} \Sigma^+).$$

We can see that φ realizes the SERCQ given in Example 2.2.

Recall that synchronized SERCQs consist of RGX-formulas that do not have variables within sub-expressions of the form $(\gamma_1 \vee \gamma_2)$. As we observe in the following result, a synchronized SERCQ can be efficiently translated into an equivalent FC[REG]-CQ.

▶ **Lemma 3.6.** *Given a synchronized* SERCQ P*, we can construct in polynomial time an* FC[REG]-CQ *that realizes* P*.*

The proof of Lemma 3.6 follows from [14, 11, 10]. The converse of Lemma 3.6 follows directly from [14]. However, one would need to define how FC[REG]-CQ-formulas can be realized by regex formulas closed under spanner algebra (details on this can be found in [10, 14]). We omit such a result as it is not the focus on this work.

In this section, we have introduced FC[REG]-CQs, and shown an efficient conversion from synchronized SERCQs to FC[REG]-CQs. Therefore, while the present paper mainly considers a tractable fragment of FC[REG]-CQ, this tractability carries over to a subclass of SERCQs.

4 Acyclic Pattern Decomposition

This section examines decomposing terminal-free patterns (i. e., patterns $\alpha \in \Xi^+$) into acyclic 2FC-CQs, where 2FC-CQ denotes the set of FC-CQs where each word equation has a right-hand side of at most length two. Patterns are the basis for FC-CQ atoms, and hence, this section gives us a foundation on which to investigate the decomposition of FC-CQs. We do not consider regular constraints, or patterns with terminals. This is because regular constraints are unary predicates, and therefore can be easily added to a join tree; and terminals can be expressed through regular constraints. We use 2FC-CQs for two reasons. Firstly, binary concatenation is the most elementary form of concatenation, as it cannot be decomposed into further (non-trivial) concatenations. Secondly, this ensures that each word equation has very low width, and therefore we can store the tables in linear space and enumerate them with constant delay – as shown in the following.

▶ **Proposition 4.1.** *Given $w \in \Sigma^*$, we can construct a data structure in linear time that, for $x, y, z \in \Xi$, allow us to enumerate $[\![x \doteq y \cdot z]\!](w)$ with constant-delay, and to decide in constant time if $\sigma \in [\![x \doteq y \cdot z]\!](w)$ holds.*

Although the cardinality of $[\![x \doteq y \cdot z]\!](w)$ is cubic in $|w|$, Proposition 4.1 allows us to represent this relation in linear space. As we can query such relations in constant time, they behave "nicer" than relations in relational algebra. Furthermore, after materializing the relations defined by each atom of an 2FC-CQ, Proposition 4.1 allows us to treat the 2FC-CQ as a relational conjunctive query. We now introduce a way to *decompose* a pattern into a conjunction of word equations where the right hand side of each atom is at most length two. We start by looking at a canonical way to decompose terminal-free patterns.

Let $\alpha \in \Xi^+$ be a terminal-free pattern. To decompose α, first we factorize α so that it can be written using only binary concatenation We define BPat, the set of all *well-bracketed patterns*, recursively as follows:

▶ **Definition 4.2.** $x \in \mathsf{BPat}$ *for all* $x \in \Xi$*, and if* $\tilde{\alpha}, \tilde{\beta} \in \mathsf{BPat}$*, then* $(\tilde{\alpha} \cdot \tilde{\beta}) \in \mathsf{BPat}$.[2]

We extend the notion of a factor to a *sub-bracketing*. We write $\tilde{\alpha} \sqsubseteq \tilde{\beta}$ if $\tilde{\alpha}$ is a factor of $\tilde{\beta}$ and $\tilde{\alpha}, \tilde{\beta} \in \mathsf{BPat}$. Let $\alpha \in \Xi^+$, by $\mathsf{BPat}(\alpha)$ we denote the set of all bracketings which correspond to the pattern α (i. e., if we remove the brackets, then the resulting pattern is α). Every $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ can be converted into an equivalent formula $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ using the following.

▶ **Definition 4.3.** *While there exists $\tilde{\beta} \sqsubseteq \tilde{\alpha}$ where $\tilde{\beta} = (x \cdot y)$ for some $x, y \in \Xi$, we replace every occurrence of $\tilde{\beta}$ in $\tilde{\alpha}$ with a new, unique variable $z \in \Xi \setminus \mathsf{var}(\alpha)$ and add the word equation $(z \doteq x \cdot y)$ to $\Psi_{\tilde{\alpha}}$. When $\tilde{\alpha} = \tilde{\beta}$, we have that $z = \mathfrak{u}$.*

Therefore, up to renaming of variables, every $\tilde{\alpha} \in \mathsf{BPat}$ has a corresponding formula $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ. We call $\Psi_{\tilde{\alpha}}$ the *decomposition* of $\tilde{\alpha}$. The decomposition can be thought of as a logic formula expressing a *straight-line program* of the pattern (see [17] for a survey on algorithms for SLPs). We now give an example of *decomposing* a bracketing.

▶ **Example 4.4.** Let $\alpha := x_1x_2x_1x_1x_2$ and let $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ be defined as follows:

$$\tilde{\alpha} := (((x_1 \cdot x_2) \cdot x_1) \cdot (x_1 \cdot x_2)).$$

We now list $\tilde{\alpha}$ after every sub-bracketing is replaced with a variable. We also give the corresponding word equation that is added to $\Psi_{\tilde{\alpha}}$.

$(((\underline{(x_1 \cdot x_2)} \cdot x_1) \cdot \underline{(x_1 \cdot x_2)})$	$z_1 \doteq x_1 \cdot x_2$
$(\underline{(z_1 \cdot x_1)} \cdot z_1)$	$z_2 \doteq z_1 \cdot x_1$
$\underline{(z_2 \cdot z_1)}$	$\mathfrak{u} \doteq z_2 \cdot z_1$

Therefore, we get the decomposition $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ, which is defined as

$$\Psi_{\tilde{\alpha}} := \mathsf{Ans}() \leftarrow (z_1 \doteq x_1 \cdot x_2) \wedge (z_2 \doteq z_1 \cdot x_1) \wedge (\mathfrak{u} \doteq z_2 \cdot z_1).$$

Notice that every sub-bracketing of $\tilde{\alpha}$ has a corresponding word equation in $\Psi_{\tilde{\alpha}}$.

The decomposition of $\tilde{\alpha}$ is somewhat similar to the *Tseytin transformations*, see [20], which transforms a propositional logic formula into a formula in *Tseytin normal form*.

Our next focus is to study which patterns can be decomposed into an *acyclic* 2FC-CQ.

[2] For convenience, we tend use $\tilde{\alpha}$ to denote a bracketing of the pattern $\alpha \in \Xi^+$.

▶ **Definition 4.5** (Join Tree). *A* join tree *for* $\Psi \in$ 2FC-CQ *with body* $\bigwedge_{i=1}^{n} \chi_i$ *is an undirected tree* $T := (V, E)$*, where* $V := \{\chi_i \mid i \in [n]\}$*, and for all* $\chi_i, \chi_j \in V$*, if* $x \in \mathsf{var}(\chi_i)$ *and* $x \in \mathsf{var}(\chi_j)$*, then* x *appears in all nodes that lie on the path between* χ_i *and* χ_j *in* T*.*

Note that we use χ (with indices) to denote atoms of a 2FC-CQ to distinguish them from word equations with arbitrarily large right-hand sides – which we denote by η (with indices). We call $\Psi \in$ 2FC-CQ *acyclic* if there exists a join tree for Ψ. Otherwise, we call Ψ *cyclic*.

▶ **Definition 4.6** (Acyclic Patterns). *If* $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ *is a decomposition of* $\tilde{\alpha} \in \mathsf{BPat}$ *and* $\Psi_{\tilde{\alpha}}$ *is acyclic, then we call* $\tilde{\alpha}$ acyclic*. If* $\Psi_{\tilde{\alpha}}$ *is cyclic, then we call* $\tilde{\alpha}$ cyclic*. If there exists* $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ *which is acyclic, then we say that* α *is* acyclic*. Otherwise,* α *is* cyclic*.*

When determining whether a decomposition $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ is acyclic, we treat each word equation (atom) of $\Psi_{\tilde{\alpha}}$ as a single relational symbol. We also consider $\mathfrak{u}$ to be a constant symbol, since $\sigma(\mathfrak{u}) = w$ always holds. This raises the question as to whether every pattern has an acyclic decomposition. The answers is no, as the following result shows.

▶ **Proposition 4.7.** *$x_1x_2x_1x_3x_1$ is a cyclic pattern, and $x_1x_2x_3x_1$ is an acyclic pattern that has a cyclic bracketing.*

This leads to the following question: *Can we decide whether a pattern is acyclic in polynomial time?* Given a pattern $\alpha \in \Xi^+$, we have that $|\mathsf{BPat}(\alpha)| = C_{|\alpha|-1}$, where C_i is the i^{th} *Catalan number*, see [18]. As the Catalan numbers grow exponentially, a straightforward enumeration of bracketings to finding an acyclic bracketing is not enough.

If $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ is a decomposition of $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$, then we call the variable $x \in \Xi$ which represents the whole pattern the *root variable*. If x is the root variable, then the atom $(x \doteq y \cdot z)$ for some $y, z \in \Xi$, is called the *root atom*. So far, the root variable has always been $\mathfrak{u}$. In Section 5, different root variables will be considered.

Let $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ be the decomposition of $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$, where $\alpha \in \Xi^+$. We define the *concatenation tree* of $\Psi_{\tilde{\alpha}}$ as a rooted, undirected, binary tree $\mathcal{T} := (\mathcal{V}, \mathcal{E}, <, \Gamma, \tau, v_r)$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of undirected edges. If v and v' have a shared parent node, then we use $v < v'$ to denote that v is the left child and v' is the right child of their shared parent. We also have $\Gamma := \mathsf{var}(\Psi_{\tilde{\alpha}})$ and the function $\tau \colon \mathcal{V} \to \Gamma$ that *labels* nodes from the concatenation tree with variables from $\mathsf{var}(\Psi_{\tilde{\alpha}})$. We use v_r to denote the root of $\mathcal{T}$. The *concatenation tree* of $\Psi_{\tilde{\alpha}}$ is defined as follows.

▶ **Definition 4.8.** *Let* $\Psi_{\tilde{\alpha}} := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n}(z_i \doteq x_i \cdot x'_i)$ *be a decomposition of* $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$*. We carry out the construction of a concatenation tree in two steps. First, we build a tree recursively. If* $v \in \mathcal{V}$ *is labeled with* z_i *for* $i \in [n]$*, then there exists a left and right child of* v *that are labeled with* x_i *and* x'_i *respectively.*

In the second step, we prune the result of the above construction to remove redundancies. For each set of non-leaf nodes *that share a common label, we define an ordering* $\ll$*. If* $\tau(v_i) = \tau(v_j)$ *and the distance from the root of* $\mathcal{T}$ *to* v_j *is strictly less than the distance from the root to* v_i*, then* $v_j \ll v_i$*. If* $\tau(v_i) = \tau(v_j)$ *and the distance from* v_r *to* v_i *and* v_j *is equal, then* $v_j \ll v_i$ *if and only if* v_j *appears to the* right *of* v_i*. For each set of non-leaf nodes that share a common label, all nodes other than the* $\ll$*-maximum node are called* redundant*. All descendants of redundant nodes are removed.*

Concatenation trees for 2FC-CQs can be understood as a variation of *derivation trees* for straight-line programs [17]. While the pruning may seem somewhat unnatural, the concatenation tree of a decomposition is a useful tool that we shall use in Lemma 4.11 to characterize acyclic bracketings.

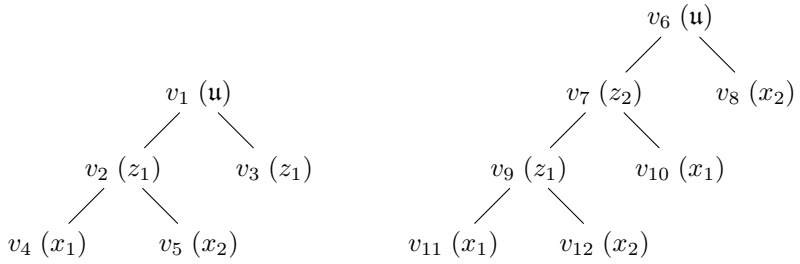


Figure 1 Concatenation trees for the decompositions of $((x_1 \cdot x_2) \cdot (x_1 \cdot x_2))$ and $(((x_1 \cdot x_2) \cdot x_1) \cdot x_2)$. This figure is used to illustrate Example 4.10.

Due to the pruning procedure, every non-leaf node represents a unique sub-bracketing. For every node v with left child v_l and right child v_r, we define $\mathsf{atom}(v) := (\tau(v) \doteq \tau(v_l) \cdot \tau(v_r))$. Note that for any two non-leaf nodes $v, v' \in \mathcal{V}$ where $v \neq v'$, we have that $\mathsf{atom}(v) \neq \mathsf{atom}(v')$. We call $v \in \mathcal{V}$ an x-parent if one of the child nodes of v is labeled x. If v is an x-parent, then $\mathsf{atom}(v)$ must contain the variable x.

▶ **Definition 4.9.** *Let $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ be the decomposition of $\tilde{\alpha} \in$ BPat and let $\mathcal{T}$ be the concatenation tree for $\Psi_{\tilde{\alpha}}$. For some $x \in \mathsf{var}(\Psi_{\tilde{\alpha}})$, we say that $\Psi_{\tilde{\alpha}}$ is* x-localized *if all nodes that exist on the path between any two x-parents in $\mathcal{T}$ are also x-parents.*

Since there is exactly one concatenation tree for a decomposition $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ of $\tilde{\alpha} \in$ BPat, we can say $\Psi_{\tilde{\alpha}}$ is x-localized without referring to the concatenation tree of $\Psi_{\tilde{\alpha}}$.

▶ **Example 4.10.** Consider the pattern $\alpha := x_1x_2x_1x_2$ and the following two bracketings:

$\tilde{\alpha}_1 := ((x_1 \cdot x_2) \cdot (x_1 \cdot x_2))$ and $\tilde{\alpha}_2 := (((x_1 \cdot x_2) \cdot x_1) \cdot x_2)$.

The bracketing $\tilde{\alpha}_1$ is decomposed into $\Psi_1 := \mathsf{Ans}() \leftarrow (z_1 \doteq x_1 \cdot x_2) \wedge (\mathfrak{u} \doteq z_1 \cdot z_1)$ and $\tilde{\alpha}_2$ is decomposed into $\Psi_2 := \mathsf{Ans}() \leftarrow (z_1 \doteq x_1 \cdot x_2) \wedge (z_2 \doteq z_1 \cdot x_1) \wedge (\mathfrak{u} \doteq z_2 \cdot x_2)$. The concatenation trees for Ψ_1 and Ψ_2 are given in Figure 1. The label for each node is given in parentheses next to the corresponding node. We can see that $\mathsf{atom}(v_2) = (z_1 \doteq x_1 \cdot x_2)$. It follows that Ψ_2 is x_1-localized, but Ψ_2 is not x_2-localized. Observe that $v_3 \ll v_2$, since v_2 appears to the left of v_3. Therefore, v_3 does not have any descendants, since it is a *redundant node.*

Utilizing concatenation trees for the decomposition $\Psi_{\tilde{\alpha}}$ of $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$, and the notion of $\Psi_{\tilde{\alpha}}$ being x-localized for $x \in \mathsf{var}(\Psi_{\tilde{\alpha}})$, we are now able to state sufficient and necessary conditions for $\alpha \in \Xi^+$ to be acyclic.

▶ **Lemma 4.11.** *The decomposition $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ of $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ is acyclic if and only if $\Psi_{\tilde{\alpha}}$ is x-localized for every $x \in \mathsf{var}(\Psi_{\tilde{\alpha}})$.*

The proof of the if-direction is rather straightforward: Take the concatenation tree of $\Psi_{\tilde{\alpha}}$, replace each non-leaf node $v \in \mathcal{V}$ with $\mathsf{atom}(v)$, then remove all leaf nodes from the concatenation tree of $\Psi_{\tilde{\alpha}}$. This gives us a join tree for $\Psi_{\tilde{\alpha}}$. The only-if direction for Lemma 4.11 is somewhat more technical. This is because we need to prove this direction for the most general join tree of $\Psi_{\tilde{\alpha}}$. We prove this by contradiction, showing that there does not exist a valid label for certain non-leaf nodes of the concatenation tree if $\Psi_{\tilde{\alpha}}$ is not x-localized for some variable $x \in \mathsf{var}(\Psi_{\tilde{\alpha}})$.

Refering back to Example 4.10, we see that Ψ_2 is not x_2-localized and therefore Ψ_2 is cyclic, whereas we have that Ψ_1 is x-localized for all $x \in \mathsf{var}(\Psi_1)$ and hence Ψ_1 is acyclic.

▶ **Theorem 4.12.** *Whether $\alpha \in \Xi^+$ is acyclic can be decided in time $\mathcal{O}(|\alpha|^7)$.*

We prove Theorem 4.12 by giving a bottom-up algorithm that continuously adds larger acyclic subpatterns of α to a set. To determine whether concatenating two acyclic subpatterns results in a larger acyclic subpattern, we also keep an edge relation and check whether x is localized, see Lemma 4.11. We terminate the algorithm when the edge relation has reached a fixed-point. In the proof of Theorem 4.12, we also show that if α is acyclic, then we can construct a concatenation tree for a decomposition for $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ in $\mathcal{O}(|\alpha|^7)$ time.

5 Acyclic FC-CQs

In this section, we generalize from decomposing patterns to decomposing FC-CQs. The main result of this section is a polynomial-time algorithm to determine whether an FC-CQ can be decomposed into an acyclic 2FC-CQ. We do this to find a notion of acyclicity for FC-CQs such that the resulting fragment is tractable.

Decomposing a word equation $(x \doteq \alpha)$ where $x \in \Xi$ and $\alpha \in (\Xi \setminus \{x\})^+$ is analogous to decomposing α, but whereas $\mathfrak{u}$ is the root variable when decomposing a pattern, we use x as the root variable when decomposing $(x \doteq \alpha)$.

If every atom of $\varphi \in$ FC-CQ is acyclic, then φ does not necessarily have tractable model checking. If this were the case, then any decomposition $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ of some $\tilde{\alpha} \in \mathsf{BPat}$ would have tractable model checking (because every word equation of the form $z \doteq x \cdot y$ is acyclic). This would imply that the membership problem for patterns can be solved in polynomial time, which contradicts [6], unless P = NP. Furthermore, if we define $\varphi \in$ FC-CQ to be acyclic if there exists a join tree for φ where every word equation is an atom, then model checking for φ is not tractable. To show this, consider $\varphi := \mathsf{Ans}() \leftarrow (\mathfrak{u} \doteq \alpha)$. Model checking for φ is equivalent to the membership problem for α, which is NP-complete [6]. Therefore, we require a more refined notion of acyclicity for FC-CQs.

In Section 4, we studied the decomposition of terminal-free patterns. If φ is an FC-CQ with the body $\mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$, then the right-hand side of some η_i may not be terminal-free. Therefore, before defining the decomposition of FC[REG]-CQs, we define a way to *normalize* FC[REG]-CQs in order to better utilize the techniques of Section 4.

▶ **Definition 5.1.** *We call an* FC-CQ *with body $\bigwedge_{i=1}^{n}(x_i \doteq \alpha_i)$* normalized *if for all $i, j \in [n]$, we have $\alpha_i \in \Xi^+$, $x_i \notin \mathsf{var}(\alpha_i)$, $\mathfrak{u} \notin \mathsf{var}(\alpha_i)$, and $\alpha_i = \alpha_j$ if and only if $i = j$.*

An FC[REG]-CQ *with body $\bigwedge_{i=1}^{n}(x_i \doteq \alpha_i) \wedge \bigwedge_{j=1}^{m}(y_j \mathrel{\dot\in} \gamma)$ is* normalized *if the subformula $\bigwedge_{i=1}^{n}(x_i \doteq \alpha_i)$ is normalized.*

Since we are interested in polynomial time algorithms, the following lemma allows us to assume that all FC-CQs are normalized without affecting any claims about complexity.

▶ **Lemma 5.2.** *Given $\varphi \in$ FC[REG]-CQ, we can construct an equivalent, normalized* FC[REG]-CQ *in time $\mathcal{O}(|\varphi|^2)$.*

To prove Lemma 5.2 we use a simple re-writing procedure. We replace every terminal factor in our formula with a new variable, and use a regular constraint to determine which terminal word that variable represents. If σ is a morphism that satisfies $(x \doteq \alpha)$ for some $\alpha \in \Xi$, then $|\sigma(x)| = |\sigma(\alpha)|$. Therefore, if $x \in \alpha$, then $|\sigma(x)| = |\sigma(\alpha_1)| + |\sigma(x)| + |\sigma(\alpha_2)|$ where $\alpha = \alpha_1 \cdot x \cdot \alpha_2$. We can then determine that $\sigma(\alpha_1) \cdot \sigma(\alpha_2) = \varepsilon$. Hence, $x \doteq \alpha$ can be replaced with $(x \doteq y) \wedge \bigwedge_{z \in \mathsf{var}(\alpha_1 \cdot \alpha_2)}(z \mathrel{\dot\in} \varepsilon)$ where y is a new and unique variable. An analogous method is used if $\mathfrak{u} \in \mathsf{var}(\alpha)$.

▶ **Example 5.3.** We define an FC[REG]-CQ along with an equivalent normalized FC[REG]-CQ:

$$\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (x_1 \doteq x_2 \cdot \mathfrak{u} \cdot x_2) \wedge (x_4 \doteq x_4) \wedge (x_3 \doteq \mathtt{aab}),$$
$$\varphi' := \mathsf{Ans}(\vec{x}) \leftarrow (\mathfrak{u} \doteq x_1) \wedge (x_2 \mathrel{\dot\in} \varepsilon) \wedge (x_4 \doteq z_2) \wedge (x_3 \doteq z_1) \wedge (z_1 \mathrel{\dot\in} \mathtt{aab}).$$

We now generalize the process of decomposing patterns to decomposing FC-CQs. For every FC-CQ $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$, we say that a 2FC-CQ $\Psi_\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \Psi_i$ is a *decomposition* of φ if every Ψ_i is a decomposition of η_i and, for all $i, j \in [n]$ with $i \neq j$, the sets of introduced variables for Ψ_i and Ψ_j are disjoint.

▶ **Example 5.4.** Let $\varphi \in$ FC-CQ be defined as follows:

$$\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (x_1 \doteq y_1 \cdot y_2 \cdot y_3) \wedge (x_2 \doteq y_2 \cdot y_3 \cdot y_3 \cdot y_4).$$

We now consider the following decompositions for each word equation of φ:

$$\Psi_1 := (x_1 \doteq y_1 \cdot z_1) \wedge (z_1 \doteq y_2 \cdot y_3), \text{ and } \Psi_2 := (x_2 \doteq z_2 \cdot y_4) \wedge (z_2 \doteq z_3 \cdot y_3) \wedge (z_3 \doteq y_2 \cdot y_3).$$

Therefore, $\Psi_\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \Psi_1 \wedge \Psi_2$ is a decomposition of φ.

▶ **Definition 5.5** (Acyclic FC-CQs). *If $\Psi_\varphi \in$ 2FC-CQ is a decomposition of $\varphi \in$ FC-CQ, we say that Ψ_φ is* acyclic *if there exists a join tree for Ψ_φ. Otherwise, Ψ_φ is* cyclic. *If there exists an acyclic decomposition of φ, then we say that φ is* acyclic. *Otherwise, φ is* cyclic.

Recall that, since $\mathfrak{u}$ is always mapped to w, we can consider $\mathfrak{u}$ a constant symbol. Therefore, if $T := (V, E)$ is a join tree for some decomposition of φ, then there can exist two nodes that both contain $\mathfrak{u}$, yet it is not necessary for all nodes on the path between these two nodes to also contain $\mathfrak{u}$. Referring back to Example 5.4, we can see that φ is acyclic by executing the GYO algorithm on the decomposition (see Chapter 6 of [1] for more information on acyclic joins). Our next focus is to study which FC-CQs are acyclic, and which are not.

▶ **Lemma 5.6.** *If $\Psi_\varphi \in$ 2FC-CQ is a decomposition of $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$, and we have a join tree $T := (V, E)$ for Ψ_φ, then we can partition T into $T^1, T^2, \ldots T^n$ such that for each $i \in [n]$, we have that T^i is a join tree for a decomposition of η_i.*

To prove Lemma 5.6, we consider a join tree $T := (V, E)$ for the acyclic decomposition $\Psi_\varphi \in$ 2FC-CQ of $\varphi \in$ FC-CQ, along with the induced subgraph of T on the set of atoms for a decomposition of a single atom of φ. We show that this subgraph is connected, and since the introduced variables are disjoint for separate atoms of φ, this forms a partition on T.

Let $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ be a normalized FC-CQ. A join tree $T := (V, E)$ for φ where $V = \{\eta_i \mid i \in [n]\}$ is called a *weak join tree*. If there exists a weak join tree for φ, then we say that φ is *weakly acyclic*. Otherwise, φ is *weakly cyclic*. Clearly weak acyclicity is not sufficient for tractability, as discussed at the start of the current section.

▶ **Example 5.7.** Consider the following normalized FC-CQ:

$$\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (\mathfrak{u} \doteq x_1 \cdot x_2 \cdot x_1 \cdot x_3 \cdot x_1) \wedge (x_1 \doteq x_4 \cdot x_5 \cdot x_5) \wedge (x_6 \doteq x_7 \cdot x_7 \cdot x_7).$$

Using the GYO algorithm, we can see that φ is weakly acyclic.

Let $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ be an FC-CQ, and let Ψ_φ be an acyclic decomposition of φ. If $T := (V, E)$ is a join tree of Ψ_φ, then for each $i \in [n]$, we use $T^i := (V^i, E^i)$ to denote the subtree of T that is a join tree for the decomposition of η_i. We know that T^i and T^j are disjoint for all $i, j \in [n]$ where $i \neq j$, see Lemma 5.6.

▶ **Lemma 5.8.** *Let* $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ *be a normalized* FC-CQ*. If any of the following conditions holds, then* φ *is cyclic:*

1. φ *is weakly cyclic,*
2. η_i *is cyclic for any* $i \in [n]$*,*
3. $|\mathsf{var}(\eta_i) \cap \mathsf{var}(\eta_j)| > 3$ *for any* $i, j \in [n]$ *where* $i \neq j$*, or*
4. $|\mathsf{var}(\eta_i) \cap \mathsf{var}(\eta_j)| = 3$*, and* $|\eta_i| > 3$ *or* $|\eta_j| > 3$ *for any* $i, j \in [n]$ *where* $i \neq j$*.*

Condition 1 can be proven by simply replacing T^i with a single node η_i for all $i \in [n]$. Condition 2 follows directly from Lemma 5.6. Conditions 3 and 4 can be proven by a contradiction: Consider the shortest path from any atom of the decomposition of η_i to any atom of the decomposition of η_j. Since the end points of these paths cannot contain all the variables that η_i and η_j share, it follows that $T := (V, E)$ is not a join tree.

While Conditions 3 and 4 might seem strict, we can pre-factor common subpatterns. For example, the conjunction $(x_1 \doteq \alpha_1 \cdot \alpha_2 \cdot \alpha_3) \wedge (x_2 \doteq \alpha_4 \cdot \alpha_2 \cdot \alpha_5)$, where $\alpha_i \in \Xi^+$ for $i \in [5]$, can be written as $(x_1 \doteq \alpha_1 \cdot z \cdot \alpha_3) \wedge (x_2 \doteq \alpha_4 \cdot z \cdot \alpha_5) \wedge (z \doteq \alpha_2)$ where $z \in \Xi$ is a new variable. We illustrate this further in the following example.

▶ **Example 5.9.** Consider the following FC-CQ:

$$\varphi := \mathsf{Ans}() \leftarrow (x_1 \doteq y_1 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5) \wedge (x_2 \doteq y_6 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5).$$

Using Lemma 5.8, we can see that φ is cyclic. However, since the right-hand side of the two word equations share a common subpattern, we can rewrite φ as

$$\varphi' := \mathsf{Ans}() \leftarrow (x_1 \doteq y_1 \cdot z) \wedge (x_2 \doteq y_6 \cdot z) \wedge (z \doteq y_2 \cdot y_3 \cdot y_4 \cdot y_5).$$

One could alter our definition of FC-CQ decomposition so that if two atoms share a bracketing, then the bracketing is replaced with the same variable (analogously to how decompositions are defined on patterns). The authors believe it is likely that such a definition of FC-CQ decomposition is equivalent to our definition of FC-CQ decomposition after "factoring out" common subpatterns between atoms.

Our next consideration is how the structure of a join tree for a decomposition of an acyclic query $\varphi \in$ FC[REG]-CQ relates to the structure of a weak join tree for φ.

▶ **Definition 5.10** (Skeleton Tree)**.** *Let* $\Psi_\varphi \in$ 2FC-CQ *be an acyclic decomposition of the query* $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$*, and let* $T := (V, E)$ *be a join tree for* Ψ_φ*. We say that a weak join tree* $T_w := (V_w, E_w)$ *is the* skeleton tree *of* T *if there exists an edge in* E *from a node in* V^i *to a node in* V^j *if and only if* $\{\eta_i, \eta_j\} \in E_w$*.*

In the proof of Lemma 5.8 (Condition 1), we show that every join tree for a decomposition has a corresponding skeleton tree. We shall leverage the fact that every join tree of a decomposition of an acyclic FC[REG]-CQ has a skeleton tree in the algorithm given in the proof of Theorem 5.14.

▶ **Example 5.11.** We define $\varphi \in$ FC-CQ and a decomposition Ψ_φ as follows:

$$\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (x_1 \doteq x_2 \cdot x_3 \cdot x_2) \wedge (x_2 \doteq x_4 \cdot x_4 \cdot x_5),$$
$$\Psi_\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (x_1 \doteq x_2 \cdot z_1) \wedge (z_1 \doteq x_3 \cdot x_2) \wedge (x_2 \doteq z_2 \cdot x_5) \wedge (z_2 \doteq x_4 \cdot x_4).$$

The skeleton tree along with the join tree of Ψ_φ are given in Figure 2.

One might assume that some skeleton trees are more "desirable" than others in terms of using it for finding an acyclic decomposition of an FC[REG]-CQ. However, as we observe next, any skeleton tree is sufficient.

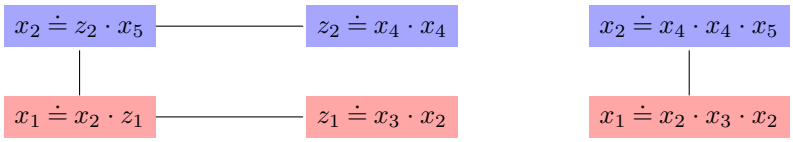


Figure 2 The join tree (left) and the skeleton tree of the join tree (right) for Example 5.11.

▶ **Lemma 5.12.** *Let $\Psi_\varphi \in$ 2FC-CQ be a decomposition of $\varphi \in$ FC-CQ. If Ψ_φ is acyclic, then any weak join tree can be used as the skeleton tree.*

Given a weak join tree of an acyclic query φ, the proof of Lemma 5.12 transforms the join tree of Ψ_φ so that the resulting join tree has the given weak join tree as its skeleton tree. Thus, we can use any weak join tree as a "template" for the eventual join tree of the decomposition (under the assumption that the query is acyclic).

While Lemma 5.8 and Lemma 5.12 give some insights and necessary conditions for deciding whether $\varphi \in$ FC-CQ is acyclic, these conditions are not sufficient. We therefore give the following lemma which is needed in the proof of Theorem 5.14 to find an acyclic decomposition of φ.

▶ **Lemma 5.13.** *Given a normalized* FC-CQ *of the form $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (z \doteq \alpha)$ and a set $C \subseteq \{\{x, y\} \mid x, y \in \mathsf{var}(z \doteq \alpha)$ and $x \neq y\}$, we can decide whether there is an acyclic decomposition $\Psi \in$ 2FC-CQ of φ such that for every $\{x, y\} \in C$, there is an atom of Ψ that contains both x and y in time $\mathcal{O}(|\alpha|^7)$.*

We prove Lemma 5.13 using a variant of the algorithm given in the proof of Theorem 4.12. The purposes of Lemma 5.13 should become clearer after giving the following necessary and sufficient criteria for an FC[REG]-CQ to be acyclic: Let $\varphi := \bigwedge_{i=1}^{m}(x_i \doteq \alpha_i) \wedge \bigwedge_{j=1}^{n}(y_j \mathrel{\dot\in} \gamma_j)$ be a normalized FC[REG]-CQ. Then, there exists an acyclic decomposition $\Psi \in$ 2FC[REG]-CQ of φ if and only if the following conditions hold:

1. φ is weakly acyclic,
2. for all $i \in [m]$ the pattern α_i is acyclic, and
3. for every $i \in [m]$, there is a decomposition Ψ_i of $x_i \doteq \alpha_i$ such that for all $j \in [m] \setminus \{i\}$ there is a decomposition Ψ_j of $x_j \doteq \alpha_j$ where there exists an atom χ_i of Ψ_i and an atom χ_j of Ψ_j that satisfies $\mathsf{var}(\chi_i) \cap \mathsf{var}(\chi_j) = \mathsf{var}(x_i \doteq \alpha_i) \cap \mathsf{var}(x_j \doteq \alpha_j)$.

We are now ready to give the main result of the paper.

▶ **Theorem 5.14.** *Whether $\varphi \in$ FC[REG]-CQ is acyclic can be decided in time $\mathcal{O}(|\varphi|^8)$.*

To prove Theorem 5.14, we first check whether $\varphi \in$ FC-CQ has any of the conditions from Lemma 5.8. If so, then we know that φ is cyclic. Then, we construct a weak join tree for φ. If there is an edge $\{\eta_i, \eta_j\}$ of the weak join tree such that η_i and η_j share exactly two variables, then we use Lemma 5.13 to decompose η_i and η_j such that there is an atom of the decomposition (of η_i and η_j), which contains the variables that η_i and η_j share. In the full proof, we show that if such decompositions do not exist, then φ is cyclic. For all other atoms of φ we can use any decomposition. The resulting acyclic decomposition is the conjunction of the decompositions of each atom. The proof of Theorem 5.14 also shows that if φ is acyclic, an acyclic decomposition can be constructed in polynomial time.

▶ **Example 5.15.** We revisit the FC[REG]-CQ that was given in the introduction:

$$\varphi := \mathsf{Ans}(x, y) \leftarrow (z \doteq z_2 \cdot x \cdot z_3 \cdot x \cdot z_4) \wedge (z \doteq z_5 \cdot y \cdot z_6) \wedge (z \mathrel{\dot\in} \gamma_{\mathsf{sen}}) \wedge (x \mathrel{\dot\in} \gamma_{\mathsf{prod}}) \wedge (y \mathrel{\dot\in} \gamma_{\mathsf{pos}}).$$

We can see this is acyclic by considering the following decomposition:

$$\Psi := \mathsf{Ans}(x,y) \leftarrow (y_1 \doteq x \cdot z_3) \wedge (y_2 \doteq y_1 \cdot x) \wedge (y_3 \doteq z_2 \cdot y_2) \wedge (z \doteq y_3 \cdot z_4) \\ \wedge (y_4 \doteq z_5 \cdot y) \wedge (z \doteq y_4 \cdot z_6) \wedge (z \mathrel{\dot\in} \gamma_{\mathsf{sen}}) \wedge (x \mathrel{\dot\in} \gamma_{\mathsf{prod}}) \wedge (y \mathrel{\dot\in} \gamma_{\mathsf{pos}}).$$

Due to the small width of the tables that each word equation of the form $(x \doteq y \cdot z)$ produces, we conclude the following:

▶ **Proposition 5.16.** *If $\Psi \in$ 2FC[REG]-CQ is acyclic, then:*

1. *Given $w \in \Sigma^*$, the model checking problem can be solved in time $\mathcal{O}(|\Psi|^2|w|^3)$.*
2. *Given $w \in \Sigma^*$, we can enumerate $[\![\Psi]\!](w)$ with $\mathcal{O}(|\Psi|^2|w|^3)$ delay.*

For FC[REG]-CQs, we first find an acyclic decomposition $\Psi_\varphi \in$ 2FC[REG]-CQ of φ in $\mathcal{O}(|\varphi|^7)$. Then, the upper bound for model checking follows from [15]. Polynomial-delay enumeration follows from [3], where it was proven that given an acyclic (relational) conjunctive query ψ and a database D, we can enumerate $\psi(D)$ with $\mathcal{O}(|\psi||D|)$ delay. Our "database" is of size $\mathcal{O}(|\varphi| \cdot |w|^3)$ as each atom of the form $(z \doteq x \cdot y)$ defines a relation of size $\mathcal{O}(|w|^3)$.

Considering techniques from [3], it may seem that the results of an acyclic FC[REG]-CQ without projections can be enumerated with constant-delay after polynomial time preprocessing. However this is not the case. New variables, that are not free, are introduced in the decomposition of φ and therefore the resulting 2FC[REG]-CQ may not be free-connex, which is required for the results of a CQ to be enumerated with constant-delay [3].

From FC[REG]-CQs to SERCQs. Combining Lemma 3.6 and Proposition 5.16 gives us a class of SERCQs for which model checking can be solved in polynomial-time, and we can enumerate results with polynomial-delay. The hardness of deciding semantic acyclicity (whether a given SERCQ can be realized by an acyclic FC[REG]-CQ) remains open. The authors believe that semantic acyclicity for SERCQs is undecidable, partly due to the fact that various minimization problems are undecidable for FC [11, 14]. For now, all we have are sufficient critiera for a SERCQ to be realized by an acyclic FC[REG]-CQ.

▶ **Definition 5.17.** *We say that a query of the form $P := \pi_Y\big(\zeta^{=}_{x_1,y_1} \cdots \zeta^{=}_{x_k,y_k}(\gamma_1 \bowtie \cdots \bowtie \gamma_n)\big)$ is* pseudo-acyclic *if for every $i \in [n]$, we have that $\gamma_i := \beta_{i_1} \cdot x_i\{\beta_{i_2}\} \cdot \beta_{i_3}$ where $x_i \in \Xi$, and where β_{i_1}, β_{i_2}, and β_{i_3} are regular expressions.*

We now show that Definition 5.17 gives sufficient criteria for an SERCQ to be realized by an acyclic FC[REG]-CQ.

▶ **Proposition 5.18.** *Given a pseudo-acyclic* SERCQ *query, we can construct in polynomial time an acyclic* FC[REG]-CQ *that realizes P.*

Freydenberger et al. [13] proved that fixing the number of atoms and the number of string equalities in a SERCQ allows for polynomial-delay enumeration of results. In contrast to this, Proposition 5.18 allows an unbounded number of joins and string equality selection operators. However, in order to have this tractability result, the expressive power of each regex formula is restricted to only allow one variable. While Proposition 5.18 gives sufficient criteria for a SERCQ to be represented by an acyclic FC[REG]-CQ, many other such classes of SERCQs likely exist. Research into finding large classes of SERCQs that map to acyclic FC[REG]-CQs seems like a promising direction for future work.

6 A Note on k-ary Decompositions

We now generalize the notion of pattern decomposition so that the length of the right-hand side of the resulting formula is less than or equal to some $k \geq 2$. While the binary decompositions might be considered the natural case, we show that generalizing to higher arities increases the expressive power of acyclic patterns. By kFC-CQ we denote the set of FC-CQ formulas that have a right-hand side of at most length k. We write BPat_k for the set of k-ary bracketed patterns over Ξ. We define BPat_k formally using the following recursive definition: For all $x \in \Xi$ we have that $x \in \mathsf{BPat}_k$, and if $\alpha_1, \alpha_2, \ldots, \alpha_i \in \mathsf{BPat}_k$ where $i \leq k$, then $(\tilde{\alpha}_1 \cdot \tilde{\alpha}_2 \cdots \tilde{\alpha}_i) \in \mathsf{BPat}_k$. We write $\tilde{\alpha} \in \mathsf{BPat}_k(\alpha)$ for some $\alpha \in \Xi^+$ if the underlying, unbracketed pattern of $\tilde{\alpha}$ is α. We can convert $\tilde{\alpha} \in \mathsf{BPat}_k$ into an equivalent kFC-CQ analogously to the binary case, see Definition 4.3.

▶ **Example 6.1.** Consider the following 4-ary bracketing:

$$\tilde{\alpha} := (((x_1 \cdot x_2 \cdot x_3) \cdot (x_4 \cdot x_2 \cdot x_4) \cdot (x_1 \cdot x_2) \cdot (x_5 \cdot x_5)) \cdot x_2).$$

As with the 2-ary case, we decompose $\tilde{\alpha}$ to get the following 4FC-CQ:

$$\Psi_{\tilde{\alpha}} := \mathsf{Ans}() \leftarrow (z_1 \doteq x_1 \cdot x_2) \wedge (z_2 \doteq x_5 \cdot x_5) \wedge (z_3 \doteq x_4 \cdot x_2 \cdot x_4) \\ \wedge (z_4 \doteq x_1 \cdot x_2 \cdot x_3) \wedge (z_5 \doteq z_4 \cdot z_3 \cdot z_1 \cdot z_2) \wedge (\mathfrak{u} \doteq z_5 \cdot x_2).$$

The definition of k-ary concatenation tree for a decomposition $\Psi_{\tilde{\alpha}} \in k$FC-CQ of $\tilde{\alpha} \in \mathsf{BPat}_k$ follows analogously to the concatenation trees for 2-ary decompositions, see Definition 4.8. The concatenation tree of the decomposition $\Psi_{\tilde{\alpha}} \in k$FC-CQ is a rooted, labeled, undirected tree $\mathcal{T} := (\mathcal{V}, \mathcal{E}, <, \Gamma, \tau, v_r)$, where $\mathcal{V}$ is the set of nodes, the relation $\mathcal{E}$ is the edge relation, and $<$ is used to denote the order of children of a node (from left to right). We have that $\Gamma := \mathsf{var}(\Psi_{\tilde{\alpha}})$ is the alphabet of labels and $\tau \colon \mathcal{V} \to \Gamma$ is the labeling function. The semantics of a k-ary concatenation tree are defined by considering the natural generalization of Definition 4.8. We say that $\Psi_{\tilde{\alpha}}$ is *x-localized* if all nodes which exist on a path between two x-parents (of $\mathcal{T}$) are also x-parents.

▶ **Proposition 6.2.** *There exists $\tilde{\alpha} \in \mathsf{BPat}_3$ such that the decomposition $\Psi \in 3$FC-CQ of $\tilde{\alpha}$ is acyclic, but there exists $x \in \mathsf{var}(\Psi)$ such that Ψ is not x-localized.*

Proof. Consider $\tilde{\alpha} := ((x_3 \cdot x_3) \cdot ((x_3 \cdot x_3) \cdot x_2) \cdot (x_1 \cdot ((x_3 \cdot x_3) \cdot x_2)))$. The bracketing $\tilde{\alpha}$ is decomposed into $\Psi_{\tilde{\alpha}} \in 3$FC-CQ, which is defined as

$$\Psi_{\tilde{\alpha}} := \mathsf{Ans}() \leftarrow (z_1 \doteq x_3 \cdot x_3) \wedge (z_2 \doteq z_1 \cdot x_2) \wedge (z_3 \doteq x_1 \cdot z_2) \wedge (\mathfrak{u} \doteq z_1 \cdot z_2 \cdot z_3).$$

The formula $\Psi_{\tilde{\alpha}}$ can be verified to be acyclic. However, $\Psi_{\tilde{\alpha}}$ is not z_1-localized. ◀

In this section, we have briefly examined k-ary decompositions, and have shown that there exists $\tilde{\alpha} \in \mathsf{BPat}_3$ such that the decomposition $\Psi \in 3$FC-CQ of $\tilde{\alpha}$ is acyclic, but Ψ is not x-localized for some $x \in \mathsf{var}(\Psi)$. The authors note that the if-direction in the proof of Lemma 4.11 implies that x-locality for all variables is a sufficient criterion for a k-ary decomposition to be acyclic. A systematic study into k-ary acyclic decompositions may yield more expressive spanners, and could be useful for pattern languages, which have been linked to FC-formulas with bounded width [14]. However, more general approaches such as bounded treewidth for binary decompositions appear to be a more promising direction for future work. Furthermore, the membership problem for a pattern α parameterized by $|\alpha|$ is W[1]-hard [8]. Since every pattern is trivially $|\alpha|$-ary acyclic, the authors believe it to be likely that the parameterized problem of model checking for k-ary acyclic decompositions is W[1]-hard.

7 Conclusions

Freydenberger and Peterfreund [14] introduced FC[REG] as a logic for querying and model checking words that behaves similar to relational FO. The present paper develops this connection further by providing a polynomial-time algorithm that either decomposes an FC[REG]-CQ into an acyclic 2FC[REG]-CQ, or determines that this is not possible. These acyclic 2FC[REG]-CQ formulas allow for polynomial-time model checking, and their results can be enumerated with polynomial-delay. Consequently, the present paper establishes a notion of tractable acyclicity for FC-CQs. Due to the close connections between FC[REG] and core spanners, this provides us with a large class of tractable SERCQs.

But this is only the first step in the study of tractable SERCQs and FC[REG]-CQs. It seems likely that more efficient algorithms for model checking and enumeration can be found by utilizing string algorithms rather than materializing the relations for each atom.

Another future direction for research is the consideration of other structural parameters, like treewidth. A systematic study of the decomposition of FC-CQs into 2FC-CQs of bounded treewidth would likely yield a large class of FC-CQs with polynomial-time model checking. As a consequence, one could define a suitable notion of treewidth for core spanners. Determining the exact class of FC-CQs with polynomial-time model checking is likely a hard problem. This is because such a result would solve the open problem in formal languages of determining exactly what patterns have polynomial-time membership.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

2 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. *ACM SIGMOD Record*, 49(1):25–32, 2020.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proceedings of CSL 2007*, pages 208–222, 2007.

4 Joachim Bremer and Dominik D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. *Information and Computation*, 220:15–43, 2012.

5 Stefan Burkhardt, Juha Kärkkäinen, and Peter Sanders. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006.

6 Andrzej Ehrenfreucht and Grzegorz Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9(2):86–88, 1979.

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, 2015.

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

9 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *Proceedings of PODS 2018*, pages 165–177, 2018.

10 Dominik D. Freydenberger. A logic for document spanners. *Theory of Computing Systems*, 63(7):1679–1754, 2019.

11 Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. *Theory of Computing Systems*, 62(4):854–898, 2018.

12 Dominik D. Freydenberger, Benny Kimelfeld, Markus Kröll, and Liat Peterfreund. Complexity bounds for relational algebra over document spanners. In *Proceedings of PODS 2019*, pages 320–334, 2019.

13 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *Proceedings of PODS 2018*, pages 137–149, 2018.

14 Dominik D. Freydenberger and Liat Peterfreund. The theory of concatenation over finite models. In *Proceedings of ICALP 2021*, pages 130:1–130:17, 2021.

15 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3):431–498, 2001.

16 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*. Cambridge University Press, 1997.

17 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups-Complexity-Cryptology*, 4(2):241–299, 2012.

18 Gloria Olive. Catalan numbers revisited. *Journal of mathematical analysis and applications*, 111(1):201–235, 1985.

19 Liat Peterfreund. Grammars for document spanners. In *Proceedings of ICDT 2021*, pages 7:1–7:18, 2021.

20 Steven David Prestwich. CNF encodings. *Handbook of satisfiability*, 185:75–97, 2009.

21 Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. In *Proceedings of ICDT 2021*, pages 4:1–4:19, 2021.

22 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of VLDB 1981*, pages 82–94, 1981.

Practical Relational Calculus Query Evaluation

Martin Raszyk
Department of Computer Science, ETH Zürich, Switzerland

David Basin
Department of Computer Science, ETH Zürich, Switzerland

Srđan Krstić
Department of Computer Science, ETH Zürich, Switzerland

Dmitriy Traytel
Department of Computer Science, University of Copenhagen, Denmark

Abstract

The relational calculus (RC) is a concise, declarative query language. However, existing RC query evaluation approaches are inefficient and often deviate from established algorithms based on finite tables used in database management systems. We devise a new translation of an arbitrary RC query into two safe-range queries, for which the finiteness of the query's evaluation result is guaranteed. Assuming an infinite domain, the two queries have the following meaning: The first is closed and characterizes the original query's relative safety, i.e., whether given a fixed database, the original query evaluates to a finite relation. The second safe-range query is equivalent to the original query, if the latter is relatively safe. We compose our translation with other, more standard ones to ultimately obtain two SQL queries. This allows us to use standard database management systems to evaluate arbitrary RC queries. We show that our translation improves the time complexity over existing approaches, which we also empirically confirm in both realistic and synthetic experiments.

2012 ACM Subject Classification Theory of computation → Database query languages (principles)

Keywords and phrases Relational calculus, relative safety, safe-range, query translation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.11

Related Version *Full Version*: `https://github.com/rc2sql/rc2sql`

1 Introduction

Codd's theorem states that all domain-independent queries of the relational calculus (RC) can be expressed in relational algebra (RA) [10]. A popular interpretation of this result is that RA suffices to express all interesting queries. This interpretation justifies why SQL evolved as the practical database query language with the RA as its mathematical foundation. SQL is declarative and abstracts over the actual RA expression used to evaluate a query. Yet, SQL's syntax inherits RA's deliberate syntactic limitations, such as union-compatibility, which ensure domain independence. RC does not have such syntactic limitations, which arguably makes it a more attractive declarative query language than both RA and SQL. The main problem of RC is that it is not immediately clear how to evaluate even domain-independent queries, much less how to handle the domain-dependent (i.e., not domain-independent) ones.

As a running example, consider a shop in which brands (unary finite relation B of brands) sell products (binary finite relation P relating brands and products) and products are reviewed by users with a score (ternary finite relation S relating products, users, and scores). We consider a brand *suspicious* if there is a user and a score such that all the brand's products were reviewed by that user with that score. An RC query computing suspicious brands is

$$Q^{susp} := \mathsf{B}(b) \wedge \exists u, s.\ \forall p.\ \mathsf{P}(b, p) \longrightarrow \mathsf{S}(p, u, s).$$

This query is domain-independent and follows closely our informal description. It is not, however, clear how to evaluate it because its second conjunct is domain-dependent as it is satisfied for every brand that does not occur in P. Finding suspicious brands using RA or SQL is a challenge, which only the best students from an undergraduate database course will accomplish. We give away an RA answer next (where $-$ is the set difference operator and $\triangleright$ is the anti-join, also known as the *generalized* difference operator [1]):

$$\pi_{brand}((\pi_{user,score}(\mathsf{S}) \times \mathsf{B}) - \pi_{brand,user,score}((\pi_{user,score}(\mathsf{S}) \times \mathsf{P}) \triangleright \mathsf{S})) \cup (\mathsf{B} - \pi_{brand}(\mathsf{P})).$$

The highlighted expressions $\pi_{user,score}(\mathsf{S})$ are called *generators*. They ensure that the left operands of the anti-join and set difference operators include or have the same columns (i.e., are union-compatible) as the corresponding right operands. (Following Codd [10], one could in principle also use the active domain to obtain canonical, but far less efficient, generators.)

Van Gelder and Topor [13, 14] present a translation from a decidable class of domain-independent RC queries, called *evaluable*, to RA expressions. Their translation of the evaluable Q^{susp} query would yield different generators, replacing both highlighted parts by $\pi_{user}(\mathsf{S}) \times \pi_{score}(\mathsf{S})$. That one can avoid this Cartesian product as shown above is subtle: Replacing only the first highlighted generator with the product results in an inequivalent RA expression.

Once we have identified suspicious brands, we may want to obtain the users whose scoring made the brands suspicious. In RC, omitting u's quantifier from Q^{susp} achieves just that:

$$Q^{susp}_{user} \coloneqq \mathsf{B}(b) \wedge \exists s.\, \forall p.\, \mathsf{P}(b,p) \longrightarrow \mathsf{S}(p,u,s).$$

In contrast, RA cannot express the same property as it is domain-dependent (hence also not evaluable and thus out of scope for Van Gelder and Topor's translation): Q^{susp}_{user} is satisfied for every user if a brand has no products, i.e., it does not occur in P. Yet, Q^{susp}_{user} is satisfied for finitely many users on every database instance where P contains at least one row for every brand from the relation B, in other words Q^{susp}_{user} is *relatively safe* on such database instances.

How does one evaluate queries that are not evaluable or even domain-dependent? The main approaches from the literature (Section 2) are either to use variants of the active domain semantics [2, 5, 15] or to abandon finite relations entirely and evaluate queries using finite representations of infinite (but well-behaved) relations such as systems of constraints [26] or automatic structures [6]. These approaches favor expressiveness over efficiency. Unlike query translations, they cannot benefit from decades of practical database research and engineering.

In this work, we translate arbitrary RC queries to RA expressions under the assumption of an infinite domain. To deal with queries that are domain-dependent, our translation produces two RA expressions, instead of a single equivalent one. The first RA expression characterizes the original RC query's relative safety, the decidable question of whether the query evaluates to a finite relation for a given database, which can be the case even for a domain-dependent query, e.g., Q^{susp}_{user}. If the original query is relatively safe on a given database, i.e., produces some finite result, then the second RA expression evaluates to the same finite result. Taken together, the two RA expressions solve the *query capturability* problem [3]: they allow us to enumerate the original RC query's finite evaluation result, or to learn that it would be infinite using RA operations on the unmodified database.

Our translation of an RC query to two RA expressions proceeds in several steps via safe-range queries and the relational algebra normal form (Section 3). We focus on the first step of translating an RC query to two safe-range RC queries (Section 4), which fundamentally differs from Van Gelder and Topor's approach and produces better generators like $\pi_{user,score}(\mathsf{S})$. Our generators strictly improve the time complexity of query evaluation (Section 4.4).

After the more standard transformations to relational algebra normal form and from there to RA expressions, we translate the resulting RA expressions into SQL using the radb tool [30]. Along the way to SQL, we leverage various ideas from the literature to optimize the overall result (Section 6). For example, we generalize Claußen et al. [9]'s approach to avoid evaluating Cartesian products like $\pi_{user,score}(\mathsf{S}) \times \mathsf{P}$ in the above translation by using count aggregations.

The overall translation allows us to use standard database management systems to evaluate RC queries. We implement our translation and use PostgreSQL to evaluate the translated queries. Using a real Amazon review dataset [23] and our synthetic benchmark that generates hard database instances for random RC queries (Section 5), we evaluate our translation's performance. The evaluation shows that our approach outperforms Van Gelder and Topor's translation (which also uses PostgreSQL for evaluation) and other approaches (Section 6).

In summary, the following are our three main contributions:

- We devise a translation of an arbitrary RC query into a pair of RA expressions as described above. The time complexity of evaluating our translation's results improves upon Van Gelder and Topor's approach [14].
- We implement our translation and extend it to produce SQL queries. The resulting tool RC2SQL makes RC a viable input language for standard database management systems. We evaluate our tool on synthetic and real data and confirm that our translation's improved asymptotic time complexity carries over into practice.
- To challenge RC2SQL (and its competitors) in our evaluation, we devise the *Data Golf* benchmark that generates hard database instances for randomly generated RC queries.

2 Related Work

We recall Trakhtenbrot's theorem and the fundamental notions of *capturability* and *data complexity*. Given an RC query over a *finite* domain, Trakhtenbrot [27] showed that it is undecidable whether there exists a (finite) structure satisfying the query. In contrast, the question of whether a fixed structure satisfies the given RC query is decidable [2].

Kifer [16] calls a query class capturable if there is an algorithm that, given a query in the class and a database instance, enumerates the query's evaluation result, i.e., all tuples satisfying the query. Avron and Hirshfeld [3] observe that Kifer's notion is restricted because it requires every query in a capturable class to be domain independent. Hence, they propose an alternative definition that we also use: A query class is capturable if there is an algorithm that, given a query in the class, a (finite or infinite) domain, and a database instance, determines whether the query's evaluation result on the database instance over the domain is finite and enumerates the result in this case. Our work solves Avron and Hirshfeld's capturability problem additionally assuming an infinite domain.

Data complexity [29] is the complexity of recognizing if a tuple satisfies a fixed query over a database, as a function of the database size. Our capturability algorithm provides an upper bound on the data complexity for RC queries over an infinite domain that have a finite evaluation result (but it cannot decide if a tuple belongs to a query's result if the result is infinite).

Next, we group related approaches to evaluating RC queries into three categories.

Structure reduction. The classical approach to handling arbitrary RC queries is to evaluate them under a finite structure [18]. The core question here is whether the evaluation produces the same result as defined by the natural semantics, which typically considers infinite domains.

Codd's theorem [10] affirmatively answers this question for domain-independent queries, restricting the structure to the *active domain*. Ailamazyan et al. [2] show that RC is a capturable query class by extending the active domain with a few additional elements, whose number depends only on the query, and evaluating the query over this finite domain. *Natural–active collapse* results [5] generalize Ailamazyan et al.'s [2] result to extensions of RC (e.g., with order relations) by combining the structure reduction with a translation-based approach. Hull and Su [15] study several semantics of RC that guarantee the finiteness of the query's evaluation result. In particular, the "output-restricted unlimited interpretation" only restricts the query's evaluation result to tuples that only contain elements in the active domain, but the quantified variables still range over the (finite or infinite) underlying domain. Our work is inspired by all these theoretical landmarks, in particular Hull and Su's work (Section 4.1). Yet we avoid using (extended) active domains, which make query evaluation impractical.

Query translation. Another strategy is to translate a given query into one that can be evaluated efficiently, for example as a sequence of RA operations. Van Gelder and Topor pioneered this approach [13,14] for RC. A core component of their translation is the choice of generators, which replace the active domain restrictions from structure reduction approaches and thereby improve the time complexity. Extensions to scalar and complex function symbols have also been studied [12, 19]. All these approaches focus on syntactic classes of RC, for which domain-independence is given, e.g., the *evaluable* queries of Van Gelder and Topor [14, Definition 5.2]. Our approach is inspired by Van Gelder and Topor's but generalizes it to handle arbitrary RC queries at the cost of assuming an infinite domain. Also, we further improve the time complexity of Van Gelder and Topor's approach by choosing better generators.

Evaluation with infinite relations. Constraint databases [26] obviate the need for using finite tables when evaluating RC queries. This yields significant expressiveness gains over RC. Yet the efficiency of the quantifier elimination procedures employed cannot compare with the simple evaluation of a projection operation in RA. Similarly, automatic structures [6] can represent the results of arbitrary RC queries finitely, but struggle with large quantities of data. We demonstrate this in our evaluation where we compare our translation to several modern incarnations of the above approaches, all based on binary decision diagrams [4, 7, 17, 20, 21].

3 Preliminaries

We introduce the RC syntax and semantics and define relevant classes of RC queries.

3.1 Relational Calculus

A signature σ is a triple $(\mathcal{C}, \mathcal{R}, \iota)$, where $\mathcal{C}$ and $\mathcal{R}$ are disjoint finite sets of constant and predicate symbols, and the function $\iota : \mathcal{R} \to \mathbb{N}$ maps each predicate symbol $r \in \mathcal{R}$ to its arity $\iota(r)$. Let $\sigma = (\mathcal{C}, \mathcal{R}, \iota)$ be a signature and $\mathcal{V}$ a countably infinite set of variables disjoint from $\mathcal{C} \cup \mathcal{R}$. The following grammar defines the syntax of RC queries:

$$Q ::= \bot \mid \top \mid x \approx t \mid r(t_1, \ldots, t_{\iota(r)}) \mid \neg Q \mid Q \vee Q \mid Q \wedge Q \mid \exists x.\, Q.$$

Here, $r \in \mathcal{R}$ is a predicate symbol, $t, t_1, \ldots, t_{\iota(r)} \in \mathcal{V} \cup \mathcal{C}$ are terms, and $x \in \mathcal{V}$ is a variable. We write $\exists \vec{v}.\, Q$ for $\exists v_1 \ldots \exists v_k.\, Q$ and $\forall \vec{v}.\, Q$ for $\neg \exists \vec{v}.\, \neg Q$, where $\vec{v}$ is a variable sequence $v_1, \ldots, v_k$. If $k = 0$, then both $\exists \vec{v}.\, Q$ and $\forall \vec{v}.\, Q$ denote just Q. Quantifiers have lower

$$
\begin{array}{lll}
(\mathcal{S},\alpha) \not\models \bot; \ (\mathcal{S},\alpha) \models \top; & & \\
(\mathcal{S},\alpha) \models r(t_1,\ldots,t_{\iota(r)}) & \text{iff} & (\alpha(t_1),\ldots,\alpha(t_{\iota(r)})) \in r^{\mathcal{S}}; \\
(\mathcal{S},\alpha) \models (Q_1 \vee Q_2) & \text{iff} & (\mathcal{S},\alpha) \models Q_1 \text{ or } (\mathcal{S},\alpha) \models Q_2; \\
(\mathcal{S},\alpha) \models (Q_1 \wedge Q_2) & \text{iff} & (\mathcal{S},\alpha) \models Q_1 \text{ and } (\mathcal{S},\alpha) \models Q_2;
\end{array}
\qquad
\begin{array}{lll}
(\mathcal{S},\alpha) \models (x \approx t) & \text{iff} & \alpha(x) = \alpha(t); \\
(\mathcal{S},\alpha) \models (\neg Q) & \text{iff} & (\mathcal{S},\alpha) \not\models Q; \\
(\mathcal{S},\alpha) \models (\exists x.\, Q) & \text{iff} & (\mathcal{S},\alpha[x \mapsto d]) \models Q, \\
 & & \text{for some } d \in \mathcal{D}.
\end{array}
$$

Figure 1 The semantics of RC.

precedence than conjunctions and disjunctions, e.g., $\exists x.\, Q_1 \wedge Q_2$ means $\exists x.\, (Q_1 \wedge Q_2)$. We use $\approx$ to denote the equality of terms in RC to distinguish it from $=$, which denotes syntactic object identity. We also write $Q_1 \longrightarrow Q_2$ for $\neg Q_1 \vee Q_2$. However, defining $Q_1 \vee Q_2$ as a shorthand for $\neg(\neg Q_1 \wedge \neg Q_2)$ would complicate later definitions, e.g., the safe-range queries (Section 3.2).

We define the subquery partial order $\sqsubseteq$ on queries inductively on the structure of RC queries, e.g., Q_1 is a subquery of the query $Q_1 \wedge \neg\exists y.\, Q_2$. One can also view $\sqsubseteq$ as the (reflexive and transitive) subterm relation on the datatype of RC queries. We denote by $\mathsf{sub}(Q)$ the set of subqueries of a query Q and by $\mathsf{fv}(Q)$ the set of *free variables* in Q. Furthermore, we denote by $\vec{\mathsf{fv}}(Q)$ the sequence of free variables in Q based on some fixed ordering of variables. We lift this notation to sets of queries in the standard way. A query Q with no free variables, i.e., $\mathsf{fv}(Q) = \emptyset$, is called *closed*. Queries of the form $r(t_1,\ldots,t_{\iota(r)})$ and $x \approx \mathsf{c}$ are called *atomic predicates*. We define the predicate $\mathsf{ap}(\cdot)$ characterizing atomic predicates, i.e., $\mathsf{ap}(Q)$ is true iff Q is an atomic predicate. Queries of the form $\exists \vec{v}.\, r(t_1,\ldots,t_{\iota(r)})$ and $\exists \vec{v}.\, x \approx \mathsf{c}$ are called *quantified predicates*. We denote by $\tilde{\exists} x.\, Q$ the query obtained by existentially quantifying a variable x from a query Q if x is free in Q, i.e., $\tilde{\exists} x.\, Q \coloneqq \exists x.\, Q$ if $x \in \mathsf{fv}(Q)$ and $\tilde{\exists} x.\, Q \coloneqq Q$ otherwise. We lift this notation to sets of queries in the standard way. We use $\tilde{\exists} x.\, Q$ (instead of $\exists x.\, Q$) when constructing a query to avoid introducing bound variables that never occur in Q.

A structure $\mathcal{S}$ over a signature $(\mathcal{C},\mathcal{R},\iota)$ consists of a non-empty domain $\mathcal{D}$ and interpretations $\mathsf{c}^{\mathcal{S}} \in \mathcal{D}$ and $r^{\mathcal{S}} \subseteq \mathcal{D}^{\iota(r)}$, for each $\mathsf{c} \in \mathcal{C}$ and $r \in \mathcal{R}$. We assume that all the relations $r^{\mathcal{S}}$ are *finite*. Note that this assumption does *not* yield a finite structure (as defined in finite model theory [18]) since the domain $\mathcal{D}$ can still be infinite. A (*variable*) *assignment* is a mapping $\alpha : \mathcal{V} \to \mathcal{D}$. We additionally define α on constant symbols $\mathsf{c} \in \mathcal{C}$ as $\alpha(\mathsf{c}) = \mathsf{c}^{\mathcal{S}}$. We write $\alpha[x \mapsto d]$ for the assignment that maps x to $d \in \mathcal{D}$ and is otherwise identical to α. We lift this notation to sequences $\vec{x}$ and $\vec{d}$ of pairwise distinct variables and arbitrary domain elements of the same length. The semantics of RC queries for a structure $\mathcal{S}$ and an assignment α is defined in Figure 1. We write $\alpha \models Q$ for $(\mathcal{S},\alpha) \models Q$ if the structure $\mathcal{S}$ is fixed in the given context. For a fixed $\mathcal{S}$, only the assignments to Q's free variables influence $\alpha \models Q$, i.e., $\alpha \models Q$ is equivalent to $\alpha' \models Q$, for every variable assignment α' that agrees with α on $\mathsf{fv}(Q)$. For closed queries Q, we write $\models Q$ and say that Q holds, since closed queries either hold for all variable assignments or for none of them. We call a finite sequence $\vec{d}$ of domain elements $d_1,\ldots d_k \in \mathcal{D}$ a *tuple*. Given a query Q and a structure $\mathcal{S}$, we denote the set of satisfying tuples for Q by

$$[\![Q]\!]^{\mathcal{S}} = \{\vec{d} \in \mathcal{D}^{|\vec{\mathsf{fv}}(Q)|} \mid \text{there exists an assignment } \alpha \text{ such that } (\mathcal{S},\alpha[\vec{\mathsf{fv}}(Q) \mapsto \vec{d}]) \models Q\}.$$

We omit $\mathcal{S}$ from $[\![Q]\!]^{\mathcal{S}}$ if $\mathcal{S}$ is fixed. We call the values from $[\![Q]\!]$ assigned to $x \in \mathsf{fv}(Q)$ column x.

The *active domain* $\mathsf{adom}^{\mathcal{S}}(Q)$ of a query Q and a structure $\mathcal{S}$ is a subset of the domain $\mathcal{D}$ containing the interpretations $\mathsf{c}^{\mathcal{S}}$ of all constant symbols that occur in Q and the values in the relations $r^{\mathcal{S}}$ interpreting all predicate symbols that occur in Q. Since $\mathcal{C}$ and $\mathcal{R}$ are finite and all $r^{\mathcal{S}}$ are finite relations of a finite arity $\iota(r)$, the active domain $\mathsf{adom}^{\mathcal{S}}(Q)$ is also a finite set. We omit $\mathcal{S}$ from $\mathsf{adom}^{\mathcal{S}}(Q)$ if $\mathcal{S}$ is fixed in the given context.

Queries Q_1 and Q_2 over the same signature are *equivalent*, written $Q_1 \equiv Q_2$, if $(\mathcal{S}, \alpha) \models Q_1 \iff (\mathcal{S}, \alpha) \models Q_2$, for every $\mathcal{S}$ and α. Queries Q_1 and Q_2 over the same signature are *inf-equivalent*, written $Q_1 \overset{\infty}{\equiv} Q_2$, if $(\mathcal{S}, \alpha) \models Q_1 \iff (\mathcal{S}, \alpha) \models Q_2$, for every $\mathcal{S}$ with an *infinite* domain $\mathcal{D}$ and every α. Clearly, equivalent queries are also inf-equivalent.

A query Q is *domain-independent* if $[\![Q]\!]^{\mathcal{S}_1} = [\![Q]\!]^{\mathcal{S}_2}$ holds for every two structures $\mathcal{S}_1$ and $\mathcal{S}_2$ that agree on the interpretations of constants ($\mathsf{c}^{\mathcal{S}_1} = \mathsf{c}^{\mathcal{S}_2}$) and predicates ($r^{\mathcal{S}_1} = r^{\mathcal{S}_2}$), while their domains $\mathcal{D}_1$ and $\mathcal{D}_2$ may differ. Agreement on the interpretations implies $\mathsf{adom}^{\mathcal{S}_1}(Q) = \mathsf{adom}^{\mathcal{S}_2}(Q) \subseteq \mathcal{D}_1 \cap \mathcal{D}_2$. It is undecidable whether an RC query is domain-independent [24, 28].

We denote by $Q[x \mapsto y]$ the query obtained from the query Q after replacing each free occurrence of the variable x by the variable y (possibly renaming bound variables to avoid capture) and performing constant propagation, i.e., simplifications like $(x \approx x) \equiv \top$, $Q \wedge \bot \equiv \bot$, $Q \vee \bot \equiv Q$, etc. We lift this notation to sets of queries in the standard way. Finally, we denote by $Q[x/\bot]$ the query obtained from Q after replacing every atomic predicate or equality containing a free variable x by $\bot$ (except for $x \approx x$) and performing constant propagation.

The function $\mathsf{flat}^{\oplus}(Q)$, where $\oplus \in \{\vee, \wedge\}$, computes a set of queries by "flattening" the operator $\oplus$: $\mathsf{flat}^{\oplus}(Q) \coloneqq \mathsf{flat}^{\oplus}(Q_1) \cup \mathsf{flat}^{\oplus}(Q_2)$ if $Q = Q_1 \oplus Q_2$ and $\mathsf{flat}^{\oplus}(Q) \coloneqq \{Q\}$ otherwise.

3.2 Safe-Range Queries

The class of *safe-range* queries [1] is a decidable subset of domain-independent RC queries. Its definition is based on the notion of range-restricted variables of a query. A variable is called *range-restricted* if "its possible values all lie within the active domain of the query" [1]. Intuitively, atomic predicates restrict the possible values of a variable that occurs in them as a term. An equality $x \approx y$ can extend the set of range-restricted variables in a conjunction $Q \wedge x \approx y$: If x or y is range-restricted in Q, then both x and y are range-restricted in $Q \wedge x \approx y$.

We formalize range-restricted variables using the *generated* relation $\mathsf{gen}(x, Q, \mathcal{G})$, defined in Figure 2. Specifically, $\mathsf{gen}(x, Q, \mathcal{G})$ holds if x is a range-restricted variable in Q and every satisfying assignment for Q satisfies some quantified predicate, referred to as *generator*, from $\mathcal{G}$. Note that, unlike in a similar definition by Van Gelder and Topor [14, Figure 5] that defines the rule $\mathsf{gen}(x, \exists y.\, Q_y, \mathcal{G})$ if $x \neq y$ and $\mathsf{gen}(x, Q_y, \mathcal{G})$, we modify the rule's conclusion to existentially quantify the bound variable y from all queries in $\mathcal{G}$ where y occurs: $\mathsf{gen}(x, \exists y.\, Q_y, \tilde{\exists} y.\, \mathcal{G})$. Hence, $\mathsf{gen}(x, Q, \mathcal{G})$ implies $\mathsf{fv}(\mathcal{G}) \subseteq \mathsf{fv}(Q)$. We now formalize these relationships.

▶ **Lemma 1.** *Let Q be a query, $x \in \mathsf{fv}(Q)$, and $\mathcal{G}$ be a set of quantified predicates such that $\mathsf{gen}(x, Q, \mathcal{G})$. Then (i) for every $Q_{qp} \in \mathcal{G}$, we have $x \in \mathsf{fv}(Q_{qp})$ and $\mathsf{fv}(Q_{qp}) \subseteq \mathsf{fv}(Q)$, (ii) for every α such that $\alpha \models Q$, there exists $Q_{qp} \in \mathcal{G}$ such that $\alpha \models Q_{qp}$, and (iii) $Q[x/\bot] = \bot$.*

▶ **Definition 2.** *We define $\mathsf{gen}(x, Q)$ to hold iff there exists a set $\mathcal{G}$ such that $\mathsf{gen}(x, Q, \mathcal{G})$. Let $\mathsf{nongens}(Q) \coloneqq \{x \in \mathsf{fv}(Q) \mid \mathsf{gen}(x, Q)$ does not hold$\}$ be the set of free variables in a query Q that are not range-restricted. A query Q has* range-restricted free variables *if every free variable of Q is range-restricted, i.e., $\mathsf{nongens}(Q) = \emptyset$. A query Q has* range-restricted bound variables *if the bound variable y in every subquery $\exists y.\, Q_y$ of Q is range-restricted, i.e., $\mathsf{gen}(y, Q_y)$ holds. A query is* safe-range *if it has range-restricted free and range-restricted bound variables.*

$\mathsf{gen}(x, \bot, \emptyset)$;
$\mathsf{gen}(x, Q, \{Q\})$ if $\mathsf{ap}(Q)$ and $x \in \mathsf{fv}(Q)$;
$\mathsf{gen}(x, \neg\neg Q, \mathcal{G})$ if $\mathsf{gen}(x, Q, \mathcal{G})$;
$\mathsf{gen}(x, \neg(Q_1 \vee Q_2), \mathcal{G})$
  if $\mathsf{gen}(x, (\neg Q_1) \wedge (\neg Q_2), \mathcal{G})$;
$\mathsf{gen}(x, \neg(Q_1 \wedge Q_2), \mathcal{G})$
  if $\mathsf{gen}(x, (\neg Q_1) \vee (\neg Q_2), \mathcal{G})$;
$\mathsf{gen}(x, Q_1 \vee Q_2, \mathcal{G}_1 \cup \mathcal{G}_2)$
  if $\mathsf{gen}(x, Q_1, \mathcal{G}_1)$ and $\mathsf{gen}(x, Q_2, \mathcal{G}_2)$;
$\mathsf{gen}(x, Q_1 \wedge Q_2, \mathcal{G})$
  if $\mathsf{gen}(x, Q_1, \mathcal{G})$ or $\mathsf{gen}(x, Q_2, \mathcal{G})$;
$\mathsf{gen}(x, Q \wedge x \approx y, \mathcal{G}[y \mapsto x])$
  if $\mathsf{gen}(y, Q, \mathcal{G})$;
$\mathsf{gen}(x, Q \wedge y \approx x, \mathcal{G}[y \mapsto x])$
  if $\mathsf{gen}(y, Q, \mathcal{G})$;
$\mathsf{gen}(x, \exists y.\, Q_y, \tilde{\exists} y.\, \mathcal{G})$
  if $x \neq y$ and $\mathsf{gen}(x, Q_y, \mathcal{G})$.

Figure 2 The *generated* relation.

$\mathsf{cov}(x, x \approx x, \emptyset)$;
$\mathsf{cov}(x, Q, \emptyset)$ if $x \notin \mathsf{fv}(Q)$;
$\mathsf{cov}(x, x \approx y, \{x \approx y\})$ if $x \neq y$;
$\mathsf{cov}(x, y \approx x, \{x \approx y\})$ if $x \neq y$;
$\mathsf{cov}(x, Q, \{Q\})$ if $\mathsf{ap}(Q)$ and $x \in \mathsf{fv}(Q)$;
$\mathsf{cov}(x, \neg Q, \mathcal{G})$ if $\mathsf{cov}(x, Q, \mathcal{G})$;
$\mathsf{cov}(x, Q_1 \vee Q_2, \mathcal{G}_1 \cup \mathcal{G}_2)$ if $\mathsf{cov}(x, Q_1, \mathcal{G}_1)$ and $\mathsf{cov}(x, Q_2, \mathcal{G}_2)$;
$\mathsf{cov}(x, Q_1 \vee Q_2, \mathcal{G})$ if $\mathsf{cov}(x, Q_1, \mathcal{G})$ and $Q_1[x/\bot] = \top$;
$\mathsf{cov}(x, Q_1 \vee Q_2, \mathcal{G})$ if $\mathsf{cov}(x, Q_2, \mathcal{G})$ and $Q_2[x/\bot] = \top$;
$\mathsf{cov}(x, Q_1 \wedge Q_2, \mathcal{G}_1 \cup \mathcal{G}_2)$ if $\mathsf{cov}(x, Q_1, \mathcal{G}_1)$ and $\mathsf{cov}(x, Q_2, \mathcal{G}_2)$;
$\mathsf{cov}(x, Q_1 \wedge Q_2, \mathcal{G})$ if $\mathsf{cov}(x, Q_1, \mathcal{G})$ and $Q_1[x/\bot] = \bot$;
$\mathsf{cov}(x, Q_1 \wedge Q_2, \mathcal{G})$ if $\mathsf{cov}(x, Q_2, \mathcal{G})$ and $Q_2[x/\bot] = \bot$;
$\mathsf{cov}(x, \exists y.\, Q_y, \tilde{\exists} y.\, \mathcal{G})$
  if $x \neq y$ and $\mathsf{cov}(x, Q_y, \mathcal{G})$ and $(x \approx y) \notin \mathcal{G}$;
$\mathsf{cov}(x, \exists y.\, Q_y, \tilde{\exists} y.\, \mathcal{G} \setminus \{x \approx y\} \cup \mathcal{G}_y[y \mapsto x])$
  if $x \neq y$ and $\mathsf{cov}(x, Q_y, \mathcal{G})$ and $\mathsf{gen}(y, Q_y, \mathcal{G}_y)$.

Figure 3 The *covered* relation.

Relational algebra normal form (RANF) is a class of safe-range queries that can be easily mapped to RA and evaluated using the RA operations for projection, column duplication, selection, set union, binary join, and anti-join. Following a standard textbook [1, Section 5.4], we define the predicate $\mathsf{ranf}(\cdot)$ characterizing RANF queries and the translation $\mathsf{sr2ranf}(\cdot)$ of a safe-range query into an equivalent RANF query.

3.3 Query Cost

To assess the time complexity of evaluating a RANF query Q, we define the *cost* of Q over a structure $\mathcal{S}$, denoted $\mathsf{cost}^{\mathcal{S}}(Q)$, to be the sum of intermediate result sizes over all RANF subqueries of Q. Formally, $\mathsf{cost}^{\mathcal{S}}(Q) \coloneqq \sum_{Q' \sqsubseteq Q,\ \mathsf{ranf}(Q')} \left|[\![Q']\!]^{\mathcal{S}}\right| \cdot |\mathsf{fv}(Q')|$. This corresponds to evaluating Q following its RANF structure using the RA operations. The complexity of these operations is linear in the combined input and output size (ignoring logarithmic factors due to set operations). The output size (the number of tuples times the number of variables) is counted in $\left|[\![Q']\!]^{\mathcal{S}}\right| \cdot |\mathsf{fv}(Q')|$ and the input size is counted as the output size for the input subqueries. Repeated subqueries are only considered once, which does not affect the asymptotics of query cost. In practice, the evaluation results for common subqueries can be reused.

4 Query Translation

Our approach to evaluating an arbitrary RC query Q over a fixed structure $\mathcal{S}$ with an infinite domain $\mathcal{D}$ proceeds by translating Q into a pair of safe-range queries (Q_{fin}, Q_{inf}) such that
(fv) $\mathsf{fv}(Q_{fin}) = \mathsf{fv}(Q)$ unless Q_{fin} is syntactically equal to $\bot$; $\mathsf{fv}(Q_{inf}) = \emptyset$;
(eval) $[\![Q]\!]$ is an infinite set if Q_{inf} holds; otherwise $[\![Q]\!] = [\![Q_{fin}]\!]$ is a finite set.
Since the queries Q_{fin} and Q_{inf} are safe-range, they are domain-independent and thus $[\![Q_{fin}]\!]$ is a finite set of tuples. In particular, $[\![Q]\!]$ is a finite set of tuples if Q_{inf} does not hold. Our translation generalizes Hull and Su's case distinction that restricts bound variables [15] to restrict all variables. Moreover, we use Van Gelder and Topor's idea to replace the active domain by a smaller set (generator) specific to each variable [14] while further improving the generators.

4.1 Restricting One Variable

Let x be a free variable in a query $\tilde{Q}$ with range-restricted bound variables. This assumption on $\tilde{Q}$ will be established by translating an arbitrary query Q bottom-up (Section 4.2). In this section, we develop a translation of $\tilde{Q}$ into an equivalent query $\tilde{Q}'$ that satisfies the following:

- $\tilde{Q}'$ has range-restricted bound variables;
- $\tilde{Q}'$ is a disjunction and x is range-restricted in all but the last disjunct.

The disjunct in which x is not range-restricted has a special form that is central to our translation: it is the conjunction of a query in which x does not occur and a query that is satisfied by infinitely many values of x. From the case distinction "for the corresponding variable: in or out of *adom*, and equality or inequality to other 'previous' variables if out of *adom*" [15], we translate $\tilde{Q}$ into the following equivalent query:

$$\begin{aligned}\tilde{Q} \equiv (\tilde{Q} \wedge x \in \mathsf{adom}(\tilde{Q})) \vee \textstyle\bigvee_{y \in \mathsf{fv}(\tilde{Q}) \setminus \{x\}} (\tilde{Q}[x \mapsto y] \wedge x \approx y) \vee \\ (\tilde{Q}[x/\bot] \wedge \neg(x \in \mathsf{adom}(\tilde{Q}) \vee \textstyle\bigvee_{y \in \mathsf{fv}(\tilde{Q}) \setminus \{x\}} x \approx y)).\end{aligned}$$

Here, $x \in \mathsf{adom}(\tilde{Q})$ stands for an RC query with a single free variable x that is satisfied by an assignment α if and only if $\alpha(x) \in \mathsf{adom}^{\mathcal{S}}(\tilde{Q})$. The translation distinguishes the following three cases for a fixed assignment α:

- if $\alpha(x) \in \mathsf{adom}^{\mathcal{S}}(\tilde{Q})$ holds, then we do not alter the query $\tilde{Q}$;
- if $x \approx y$ holds for some free variable $y \in \mathsf{fv}(\tilde{Q}) \setminus \{x\}$, then x can be replaced by y in $\tilde{Q}$;
- otherwise, $\tilde{Q}$ is equivalent to $\tilde{Q}[x/\bot]$, i.e., all atomic predicates with a free occurrence of x can be replaced by $\bot$ (because $\alpha(x) \notin \mathsf{adom}^{\mathcal{S}}(\tilde{Q})$), all equalities $x \approx y$ and $y \approx x$ for $y \in \mathsf{fv}(\tilde{Q}) \setminus \{x\}$ can be replaced by $\bot$ (because $\alpha(x) \neq \alpha(y)$), and all equalities $x \approx z$ for a bound variable z can be replaced by $\bot$ (because $\alpha(x) \notin \mathsf{adom}^{\mathcal{S}}(\tilde{Q})$ and z is range-restricted in its subquery $\exists z.\, Q_z$, by assumption, i.e., $\mathsf{gen}(z, Q_z)$ holds and thus, for all α', we have $\alpha' \models \exists z.\, Q_z$ if and only if there exists $d \in \mathsf{adom}^{\mathcal{S}}(Q_z) \subseteq \mathsf{adom}^{\mathcal{S}}(\tilde{Q})$ such that $\alpha'[z \mapsto d] \models Q_z$).

Note that $\exists \vec{\mathsf{fv}}(Q) \setminus \{x\}.\, Q$ is the query in which all free variables of Q except x are existentially quantified. Given a set of quantified predicates $\mathcal{G}$, we write $\exists \vec{\alpha}.\, \mathcal{G}$ for $\bigvee_{Q_{qp} \in \mathcal{G}} \exists \vec{\alpha}.\, Q_{qp}$. To avoid enumerating the entire active domain $\mathsf{adom}^{\mathcal{S}}(Q)$ of the query Q and a structure $\mathcal{S}$, Van Gelder and Topor [14] replace the condition $x \in \mathsf{adom}(Q)$ in their translation by $\exists \vec{\mathsf{fv}}(\mathcal{G}) \setminus \{x\}.\, \mathcal{G}$, where generator set $\mathcal{G}$ is a subset of atomic predicates. Because their translation [14] must yield an equivalent query (for every finite or infinite domain), $\mathcal{G}$ must satisfy, for all α,

$$\begin{aligned}&\alpha \models \neg\exists \vec{\mathsf{fv}}(\mathcal{G}) \setminus \{x\}.\, \mathcal{G} \implies (\alpha \models Q \iff \alpha \models Q[x/\bot]) && (\text{VGT}_1) \quad \text{and} \\ &\alpha \models Q[x/\bot] \implies \alpha \models \forall x.\, Q && (\text{VGT}_2).\end{aligned}$$

Note that (VGT_2) does not hold for the query $Q \coloneqq \neg \mathsf{B}(x)$ and thus a generator set $\mathcal{G}$ of atomic predicates satisfying (VGT_2) only exists for a proper subset of all RC queries. In contrast, we only require that $\mathcal{G}$ satisfies (VGT_1) in our translation. To this end, we define a *covered* relation $\mathsf{cov}(x, Q, \mathcal{G})$ (in contrast to Van Gelder and Topor's *constrained* relation con [14, Figure 5]) such that, for every variable x and query $\tilde{Q}$ with range-restricted bound variables, there exists at least one set $\mathcal{G}$ such that $\mathsf{cov}(x, \tilde{Q}, \mathcal{G})$ and (VGT_1) holds. Figure 3 shows the definition of this relation. Unlike the generator set $\mathcal{G}$ in $\mathsf{gen}(x, Q, \mathcal{G})$, the *cover* set $\mathcal{G}$ in $\mathsf{cov}(x, Q, \mathcal{G})$ may also contain equalities between two variables. Hence, we define a function $\mathsf{qps}(\mathcal{G})$ that collects all *generators*, i.e., quantified predicates and a function $\mathsf{eqs}(x, \mathcal{G})$ that collects all *variables* y distinct from x occurring in equalities of the form $x \approx y$. We use $\mathsf{qps}^{\vee}(\mathcal{G})$ to denote the query $\bigvee_{Q_{qp} \in \mathsf{qps}(\mathcal{G})} Q_{qp}$. We state the soundness and completeness of the relation $\mathsf{cov}(x, Q, \mathcal{G})$ in the next lemma, which follows by induction on the derivation of $\mathsf{cov}(x, \tilde{Q}, \mathcal{G})$.

▶ **Lemma 3.** *Let $\tilde{Q}$ be a query with range-restricted bound variables, $x \in \mathsf{fv}(\tilde{Q})$. Then there exists a set $\mathcal{G}$ of quantified predicates and equalities such that $\mathsf{cov}(x, \tilde{Q}, \mathcal{G})$ holds and, for any such $\mathcal{G}$ and all α,*

$$\alpha \models \neg(\mathsf{qps}^{\vee}(\mathcal{G}) \vee \bigvee_{y \in \mathsf{eqs}(x,\mathcal{G})} x \approx y) \implies (\alpha \models \tilde{Q} \iff \alpha \models \tilde{Q}[x/\bot]).$$

Finally, to preserve the dependencies between the variable x and the remaining free variables of Q occurring in the quantified predicates from $\mathsf{qps}(\mathcal{G})$, we do not project $\mathsf{qps}(\mathcal{G})$ on the single variable x, i.e., we restrict x by $\mathsf{qps}^{\vee}(\mathcal{G})$ instead of $\exists \vec{\mathsf{fv}}(Q) \setminus \{x\}.\, \mathsf{qps}(\mathcal{G})$. From Lemma 3, we derive our optimized translation characterized by the following lemma.

▶ **Lemma 4.** *Let $\tilde{Q}$ be a query with range-restricted bound variables, $x \in \mathsf{fv}(\tilde{Q})$, and $\mathcal{G}$ be such that $\mathsf{cov}(x, \tilde{Q}, \mathcal{G})$ holds. Then $x \in \mathsf{fv}(Q_{qp})$ and $\mathsf{fv}(Q_{qp}) \subseteq \mathsf{fv}(\tilde{Q})$, for every $Q_{qp} \in \mathsf{qps}(\mathcal{G})$, and*

$$\begin{aligned}\tilde{Q} \equiv (\tilde{Q} \wedge \mathsf{qps}^{\vee}(\mathcal{G})) \vee \bigvee_{y \in \mathsf{eqs}(x,\mathcal{G})} (\tilde{Q}[x \mapsto y] \wedge x \approx y) \vee \\ (\tilde{Q}[x/\bot] \wedge \neg(\mathsf{qps}^{\vee}(\mathcal{G}) \vee \bigvee_{y \in \mathsf{eqs}(x,\mathcal{G})} x \approx y)).\end{aligned} \tag{★}$$

Note that x is not guaranteed to be range-restricted in (★)'s last disjunct. However, it occurs only in the negation of a disjunction of quantified predicates with a free occurrence of x and equalities of the form $x \approx \mathsf{c}$ or $x \approx y$. We will show how to handle such occurrences in Sections 4.2 and 4.3. Moreover, the negation of the disjunction can be omitted if (VGT_2) holds.

4.2 Restricting Bound Variables

Let x be a free variable in a query $\tilde{Q}$ with range-restricted bound variables. Suppose that the variable x is not range-restricted, i.e., $\mathsf{gen}(x, \tilde{Q})$ does not hold. To translate $\exists x.\, \tilde{Q}$ into an inf-equivalent query with range-restricted bound variables ($\exists x.\, \tilde{Q}$ does not have range-restricted bound variables precisely because x is not range-restricted in $\tilde{Q}$), we first apply (★) to $\tilde{Q}$ and distribute the existential quantifier binding x over disjunction. Next we observe that

$$\exists x.\, (\tilde{Q}[x \mapsto y] \wedge x \approx y) \equiv \tilde{Q}[x \mapsto y] \wedge \exists x.\, (x \approx y) \equiv \tilde{Q}[x \mapsto y],$$

where the first equivalence follows because x does not occur free in $\tilde{Q}[x \mapsto y]$ and the second equivalence follows from the straightforward validity of $\exists x.\, (x \approx y)$. Moreover, we observe that

$$\exists x.\, (\tilde{Q}[x/\bot] \wedge \neg(\mathsf{qps}^{\vee}(\mathcal{G}) \vee \bigvee_{y \in \mathsf{eqs}(x,\mathcal{G})} x \approx y)) \overset{\infty}{\equiv} \tilde{Q}[x/\bot]$$

because x is not free in $\tilde{Q}[x/\bot]$ and there exists a value d for x in the infinite domain $\mathcal{D}$ such that $x \neq y$ holds for all finitely many $y \in \mathsf{eqs}(x, \mathcal{G})$ and d is not among the finitely many values interpreting the quantified predicates in $\mathsf{qps}(\mathcal{G})$. Altogether, we obtain the following lemma.

▶ **Lemma 5.** *Let $\tilde{Q}$ be a query with range-restricted bound variables, $x \in \mathsf{fv}(\tilde{Q})$, and $\mathcal{G}$ be a set of quantified predicates and equalities such that $\mathsf{cov}(x, \tilde{Q}, \mathcal{G})$ holds. Then*

$$\exists x.\, \tilde{Q} \overset{\infty}{\equiv} (\exists x.\, \tilde{Q} \wedge \mathsf{qps}^{\vee}(\mathcal{G})) \vee \bigvee_{y \in \mathsf{eqs}(x,\mathcal{G})} (\tilde{Q}[x \mapsto y]) \vee \tilde{Q}[x/\bot]. \tag{★∃}$$

Algorithm 1 Restricting bound variables.

input: An RC query Q.
output: A query $\tilde{Q}$ with range-restricted bound variables such that $Q \overset{\infty}{\equiv} \tilde{Q}$.

1 **function** $\mathsf{fixbound}(\mathcal{Q}, x) =$
  $\{Q_{fix} \in \mathcal{Q} \mid x \in \mathsf{nongens}(Q_{fix})\}$;
2 **function** $\mathsf{rb}(Q) =$
3 **switch** Q **do**
4 **case** $\neg Q'$ **do return** $\neg\mathsf{rb}(Q')$;
5 **case** $Q'_1 \vee Q'_2$ **do return** $\mathsf{rb}(Q'_1) \vee \mathsf{rb}(Q'_2)$;
6 **case** $Q'_1 \wedge Q'_2$ **do return** $\mathsf{rb}(Q'_1) \wedge \mathsf{rb}(Q'_2)$;
7 **case** $\exists x.\, Q_x$ **do**
8 $\mathcal{Q} \coloneqq \mathsf{flat}^{\vee}(\mathsf{rb}(Q_x))$;
9 **while** $\mathsf{fixbound}(\mathcal{Q}, x) \neq \emptyset$ **do**
10 $Q_{fix} \leftarrow \mathsf{fixbound}(\mathcal{Q}, x)$;
11 $\mathcal{G} \leftarrow \{\mathcal{G} \mid \mathsf{cov}(x, Q_{fix}, \mathcal{G})\}$;
12 $\mathcal{Q} \coloneqq (\mathcal{Q} \setminus \{Q_{fix}\}) \cup \{Q_{fix} \wedge \mathsf{qps}^{\vee}(\mathcal{G})\} \cup \bigcup_{y \in \mathsf{eqs}(x,\mathcal{G})} \{Q_{fix}[x \mapsto y]\} \cup \{Q_{fix}[x/\bot]\}$;
13 **return** $\bigvee_{\tilde{Q} \in \mathcal{Q}} \tilde{\exists} x.\, \tilde{Q}$;
14 **otherwise do return** Q;

Algorithm 2 Restricting free variables.

input: An RC query Q.
output: Safe-range query pair (Q_{fin}, Q_{inf}) for which **(fv)** and **(eval)** hold.

1 **function** $\mathsf{fixfree}(\mathcal{Q}_{fin}) =$
  $\{(Q_{fix}, Q^{=}) \in \mathcal{Q}_{fin} \mid \mathsf{nongens}(Q_{fix}) \neq \emptyset\}$;
2 **function** $\mathsf{inf}(\mathcal{Q}_{fin}, Q) = \{(Q_{\not\infty}, Q^{=}) \in \mathcal{Q}_{fin} \mid \mathsf{disjointvars}(Q_{\not\infty}, Q^{=}) \neq \emptyset \vee \mathsf{fv}(Q_{\not\infty} \wedge Q^{=}) \neq \mathsf{fv}(Q)\}$;
3 **function** $\mathsf{split}(Q) =$
4 $\mathcal{Q}_{fin} \coloneqq \{(\mathsf{rb}(Q), \top)\}$; $\mathcal{Q}_{inf} \coloneqq \emptyset$;
5 **while** $\mathsf{fixfree}(\mathcal{Q}_{fin}) \neq \emptyset$ **do**
6 $(Q_{fix}, Q^{=}) \leftarrow \mathsf{fixfree}(\mathcal{Q}_{fin})$;
7 $x \leftarrow \mathsf{nongens}(Q_{fix})$;
8 $\mathcal{G} \leftarrow \{\mathcal{G} \mid \mathsf{cov}(x, Q_{fix}, \mathcal{G})\}$;
9 $\mathcal{Q}_{fin} \coloneqq (\mathcal{Q}_{fin} \setminus \{(Q_{fix}, Q^{=})\}) \cup \{(Q_{fix} \wedge \mathsf{qps}^{\vee}(\mathcal{G}), Q^{=})\} \cup \bigcup_{y \in \mathsf{eqs}(x,\mathcal{G})} \{(Q_{fix}[x \mapsto y], Q^{=} \wedge x \approx y)\}$;
10 $\mathcal{Q}_{inf} \coloneqq \mathcal{Q}_{inf} \cup \{Q_{fix}[x/\bot]\}$;
11 **while** $\mathsf{inf}(\mathcal{Q}_{fin}, Q) \neq \emptyset$ **do**
12 $(Q_{\not\infty}, Q^{=}) \leftarrow \mathsf{inf}(\mathcal{Q}_{fin}, Q)$;
13 $\mathcal{Q}_{fin} \coloneqq \mathcal{Q}_{fin} \setminus \{(Q_{\not\infty}, Q^{=})\}$;
14 $\mathcal{Q}_{inf} \coloneqq \mathcal{Q}_{inf} \cup \{Q_{\not\infty} \wedge Q^{=}\}$;
15 **return** $(\bigvee_{(Q_{\not\infty}, Q^{=}) \in \mathcal{Q}_{fin}} (Q_{\not\infty} \wedge Q^{=}),$
  $\mathsf{rb}(\bigvee_{Q_\infty \in \mathcal{Q}_{inf}} \exists \vec{\mathsf{fv}}(Q_\infty).\, Q_\infty))$;

Our approach for restricting all bound variables recursively applies Lemma 5. Because the set $\mathcal{G}$ such that $\mathsf{cov}(x, Q, \mathcal{G})$ holds is not necessarily unique, we introduce the following (general) notation. We denote the non-deterministic choice of an object X from a non-empty set $\mathcal{X}$ as $X \leftarrow \mathcal{X}$. We define the recursive function $\mathsf{rb}(Q)$ in Algorithm 1, where rb stands for *range-restrict bound* (variables). The function converts an arbitrary RC query Q into an inf-equivalent query with range-restricted bound variables. We proceed by describing the case $\exists x.\, Q_x$. First, $\mathsf{rb}(Q_x)$ is recursively applied on Line 8 to establish the precondition of Lemma 5 that the translated query has range-restricted bound variables. Because existential quantification distributes over disjunction, we flatten disjunction in $\mathsf{rb}(Q_x)$ and process the individual disjuncts independently. We apply (★∃) to every disjunct Q_{fix} in which the variable x is not already range-restricted. For every Q'_{fix} added to $\mathcal{Q}$ after applying (★∃) to Q_{fix} the variable x is either range-restricted or does not occur in Q'_{fix}, i.e., $x \notin \mathsf{nongens}(Q'_{fix})$. This entails the termination of the loop on Lines 9–12.

▶ **Example 6.** Consider the query $Q^{susp}_{user} \coloneqq \mathsf{B}(b) \wedge \exists s.\, \forall p.\, \mathsf{P}(b,p) \longrightarrow \mathsf{S}(p,u,s)$ from Section 1. Restricting its bound variables yields the query

$$\mathsf{rb}(Q^{susp}_{user}) = \mathsf{B}(b) \wedge ((\exists s.\, (\neg\exists p.\, \mathsf{P}(b,p) \wedge \neg\mathsf{S}(p,u,s)) \wedge (\exists p.\, \mathsf{S}(p,u,s))) \vee (\neg\exists p.\, \mathsf{P}(b,p))).$$

The bound variable p is already range-restricted in Q^{susp}_{user} and thus only s must be restricted. Applying (★) to restrict s in $\neg\exists p.\, \mathsf{P}(b,p) \wedge \neg\mathsf{S}(p,u,s)$, then existentially quantifying s, and distributing the existential over disjunction yields the first disjunct in $\mathsf{rb}(Q^{susp}_{user})$ above and $\exists s.\, (\neg\exists p.\, \mathsf{P}(b,p)) \wedge \neg(\exists p.\, \mathsf{S}(p,u,s))$ as the second disjunct. Because there exists some value in the infinite domain $\mathcal{D}$ that does not belong to the finite interpretation of the atomic predicate $\mathsf{S}(p,u,s)$, the query $\exists s.\, \neg(\exists p.\, \mathsf{S}(p,u,s))$ is a tautology over $\mathcal{D}$. Hence, $\exists s.\, (\neg\exists p.\, \mathsf{P}(b,p)) \wedge \neg(\exists p.\, \mathsf{S}(p,u,s))$ is inf-equivalent to $\neg\exists p.\, \mathsf{P}(b,p)$, i.e., the second disjunct in $\mathsf{rb}(Q^{susp}_{user})$. This reasoning justifies applying (★∃) to restrict s in $\exists s.\, \neg\exists p.\, \mathsf{P}(b,p) \wedge \neg\mathsf{S}(p,u,s)$.

4.3 Restricting Free Variables

Given an arbitrary query Q, we translate the inf-equivalent query $\mathsf{rb}(Q)$ with range-restricted bound variables into a pair of safe-range queries (Q_{fin}, Q_{inf}) such that our translation's main properties **(fv)** and **(eval)** hold. Our translation is based on the following lemma.

▶ **Lemma 7.** *Let a structure $\mathcal{S}$ with an infinite domain $\mathcal{D}$ be fixed. Let x be a free variable in a query $\tilde{Q}$ with range-restricted bound variables and let $\mathsf{cov}(x, \tilde{Q}, \mathcal{G})$ for a set of quantified predicates and equalities $\mathcal{G}$. If $\tilde{Q}[x/\bot]$ is not satisfied by any tuple, then*

$$[\![\tilde{Q}]\!] = \left[\!\!\left[(\tilde{Q} \wedge \mathsf{qps}^{\vee}(\mathcal{G})) \vee \bigvee\nolimits_{y \in \mathsf{eqs}(x,\mathcal{G})} (\tilde{Q}[x \mapsto y] \wedge x \approx y)\right]\!\!\right]. \tag{☆}$$

If $\tilde{Q}[x/\bot]$ is satisfied by some tuple, then $[\![\tilde{Q}]\!]$ is an infinite set.

Proof. If $\tilde{Q}[x/\bot]$ is not satisfied by any tuple, then (☆) follows from (★). If $\tilde{Q}[x/\bot]$ is satisfied by some tuple, then the last disjunct in (★) applied to $\tilde{Q}$ is satisfied by infinitely many tuples obtained by assigning x some value from the infinite domain $\mathcal{D}$ such that $x \neq y$ holds for all finitely many $y \in \mathsf{eqs}(x, \mathcal{G})$ and x does not appear among the finitely many values interpreting the quantified predicates from $\mathsf{qps}(\mathcal{G})$. ◀

We remark that $[\![\tilde{Q}]\!]$ might be an infinite set of tuples even if $\tilde{Q}[x/\bot]$ is never satisfied, for some x. This is because $\tilde{Q}[y/\bot]$ might be satisfied by some tuple, for some y, in which case Lemma 7 (for y) implies that $[\![\tilde{Q}]\!]$ is an infinite set of tuples. Still, (☆) can be applied to $\tilde{Q}$ for x resulting in an equivalent query that is also satisfied by an infinite set of tuples.

Our approach is implemented by the function $\mathsf{split}(Q)$ defined in Algorithm 2. In the following, we describe this function and informally justify its correctness, formalized by the input/output specification. In $\mathsf{split}(Q)$, we represent the queries Q_{fin} and Q_{inf} using a set $\mathcal{Q}_{fin}$ of query pairs and a set $\mathcal{Q}_{inf}$ of queries such that

$$Q_{fin} \coloneqq \bigvee\nolimits_{(Q_{\not\approx}, Q^{=}) \in \mathcal{Q}_{fin}} (Q_{\not\approx} \wedge Q^{=}), \qquad Q_{inf} \coloneqq \bigvee\nolimits_{Q_{\infty} \in \mathcal{Q}_{inf}} \exists \vec{\mathsf{fv}}(Q_{\infty}).\, Q_{\infty},$$

and, for every $(Q_{\not\approx}, Q^{=}) \in \mathcal{Q}_{fin}$, $Q^{=}$ is a conjunction of equalities. As long as there exists some $(Q_{fix}, Q^{=}) \in \mathcal{Q}_{fin}$ such that $\mathsf{nongens}(Q_{fix}) \neq \emptyset$, we apply (☆) to Q_{fix} and add the query $Q_{fix}[x/\bot]$ to $\mathcal{Q}_{inf}$. We remark that if we applied (☆) to the entire disjunct $Q_{fix} \wedge Q^{=}$, the loop on Lines 5–10 might not terminate. Note that, for every $(Q'_{fix}, Q'^{=})$ added to $\mathcal{Q}_{fin}$ after applying (☆) to Q_{fix}, $\mathsf{nongens}(Q'_{fix})$ is a proper subset of $\mathsf{nongens}(Q_{fix})$. This entails the termination of the loop on Lines 5–10. Finally, if $[\![Q_{fix}]\!]$ is an infinite set of tuples, then $[\![Q_{fix} \wedge Q^{=}]\!]$ is an infinite set of tuples, too. This is because the equalities in $Q^{=}$ merely duplicate columns of the query Q_{fix}. Hence, it indeed suffices to apply (☆) to Q_{fix} instead of $Q_{fix} \wedge Q^{=}$.

After the loop on Lines 5–10 in Algorithm 2 terminates, for every $(Q_{\not\approx}, Q^{=}) \in \mathcal{Q}_{fin}$, $Q_{\not\approx}$ is a safe-range query and $Q^{=}$ is a conjunction of equalities such that $\mathsf{fv}(Q_{\not\approx} \wedge Q^{=}) = \mathsf{fv}(Q)$. However, the query $Q_{\not\approx} \wedge Q^{=}$ need not be safe-range, e.g., if $Q_{\not\approx} \coloneqq \mathsf{B}(x)$ and $Q^{=} \coloneqq (x \approx y \wedge u \approx v)$. Given a set of equalities $\mathcal{Q}^{=}$, let $\mathsf{classes}(\mathcal{Q}^{=})$ be the set of equivalence classes of free variables $\mathsf{fv}(\mathcal{Q}^{=})$ with respect to $\mathcal{Q}^{=}$. For instance, $\mathsf{classes}(\{x \approx y, y \approx z, u \approx v\}) = \{\{x, y, z\}, \{u, v\}\}$. Let $\mathsf{disjointvars}(Q_{\not\approx}, Q^{=}) \coloneqq \bigcup_{V \in \mathsf{classes}(\mathsf{flat}^{\wedge}(Q^{=})), V \cap \mathsf{fv}(Q_{\not\approx}) = \emptyset} V$ be the set of all variables in equivalence classes from $\mathsf{classes}(\mathsf{flat}^{\wedge}(Q^{=}))$ that are disjoint from $Q_{\not\approx}$'s free variables. Then, $Q_{\not\approx} \wedge Q^{=}$ is safe-range if and only if $\mathsf{disjointvars}(Q_{\not\approx}, Q^{=}) = \emptyset$ (recall the definition of safe-range).

Now if $\mathsf{disjointvars}(Q_{\not\approx}, Q^{=}) \neq \emptyset$ and $Q_{\not\approx} \wedge Q^{=}$ is satisfied by some tuple, then $[\![Q_{\not\approx} \wedge Q^{=}]\!]$ is an infinite set of tuples because all equivalence classes of variables in $\mathsf{disjointvars}(Q_{\not\approx}, Q^{=}) \neq \emptyset$ can be assigned arbitrary values from the infinite domain $\mathcal{D}$. In our example with

$Q_{\not\approx} := \mathsf{B}(x)$ and $Q^{=} := (x \approx y \wedge u \approx v)$, we have $\mathsf{disjointvars}(Q_{\not\approx}, Q^{=}) = \{u, v\} \neq \emptyset$. Moreover, if $\mathsf{fv}(Q_{\not\approx} \wedge Q^{=}) \neq \mathsf{fv}(Q)$ and $Q_{\not\approx} \wedge Q^{=}$ is satisfied by some tuple, then this tuple can be extended to infinitely many tuples over $\mathsf{fv}(Q)$ by choosing arbitrary values from the infinite domain $\mathcal{D}$ for the variables in the non-empty set $\mathsf{fv}(Q) \setminus \mathsf{fv}(Q_{\not\approx} \wedge Q^{=})$. Hence, for every $(Q_{\not\approx}, Q^{=}) \in \mathcal{Q}_{fin}$ with $\mathsf{disjointvars}(Q_{\not\approx}, Q^{=}) \neq \emptyset$ or $\mathsf{fv}(Q_{\not\approx} \wedge Q^{=}) \neq \mathsf{fv}(Q)$, we remove $(Q_{\not\approx}, Q^{=})$ from $\mathcal{Q}_{fin}$ and add $Q_{\not\approx} \wedge Q^{=}$ to $\mathcal{Q}_{inf}$. Note that we only remove pairs from $\mathcal{Q}_{fin}$, hence, the loop on Lines 11–14 terminates. Afterwards, the query Q_{fin} is safe-range. However, the query Q_{inf} need not be safe-range. Indeed, every query $Q_\infty \in \mathcal{Q}_{inf}$ has range-restricted bound variables, but not all the free variables of Q_∞ need be range-restricted and thus the query $\exists \vec{\mathsf{fv}}(Q_\infty).\, Q_\infty$ need not be safe-range. But the query Q_{inf} is closed and thus the inf-equivalent query $\mathsf{rb}(Q_{inf})$ with range-restricted bound variables is safe-range.

▶ **Lemma 8.** *Let Q be an RC query and $\mathsf{split}(Q) = (Q_{fin}, Q_{inf})$. Then the queries Q_{fin} and Q_{inf} are safe-range; $\mathsf{fv}(Q_{fin}) = \mathsf{fv}(Q)$ unless Q_{fin} is syntactically equal to $\bot$; and $\mathsf{fv}(Q_{inf}) = \emptyset$.*

▶ **Lemma 9.** *Let a structure $\mathcal{S}$ with an infinite domain $\mathcal{D}$ be fixed. Let Q be an RC query and $\mathsf{split}(Q) = (Q_{fin}, Q_{inf})$. If $\models Q_{inf}$, then $[\![Q]\!]$ is an infinite set. Otherwise, $[\![Q]\!] = [\![Q_{fin}]\!]$ is a finite set.*

By Lemma 8, Q_{fin} is a safe-range (and thus also domain-independent) query. Hence, for a fixed structure $\mathcal{S}$, the tuples in $[\![Q_{fin}]\!]$ only contain elements in the active domain $\mathsf{adom}(Q_{fin})$, i.e., $[\![Q_{fin}]\!] = [\![Q_{fin}]\!] \cap \mathsf{adom}(Q_{fin})^{|\mathsf{fv}(Q_{fin})|}$. Our translation does not introduce new constants in Q_{fin} and thus $\mathsf{adom}(Q_{fin}) \subseteq \mathsf{adom}(Q)$. Hence, by Lemma 9, if $\not\models Q_{inf}$, then $[\![Q_{fin}]\!]$ is equal to the "output-restricted unlimited interpretation" [15] of Q, i.e., $[\![Q_{fin}]\!] = [\![Q]\!] \cap \mathsf{adom}(Q)^{|\mathsf{fv}(Q)|}$. In contrast, if $\models Q_{inf}$, then $[\![Q_{fin}]\!] = [\![Q]\!] \cap \mathsf{adom}(Q)^{|\mathsf{fv}(Q)|}$ does not necessarily hold. For instance, for $Q := \neg\mathsf{B}(x)$, our translation yields $\mathsf{split}(Q) = (\bot, \top)$. In this case, we have $Q_{inf} = \top$ and thus $\models Q_{inf}$ because $\neg\mathsf{B}(x)$ is satisfied by infinitely many tuples over an infinite domain. However, if $\mathsf{B}(x)$ is never satisfied, then $[\![Q_{fin}]\!] = \emptyset$ is not equal to $[\![Q]\!] \cap \mathsf{adom}(Q)^{|\mathsf{fv}(Q)|}$.

▶ **Example 10.** Consider the query $Q := \mathsf{B}(x) \vee \mathsf{P}(x, y)$. The variable y is not range-restricted in Q and thus $\mathsf{split}(Q)$ restricts y by a conjunction of Q with $\mathsf{P}(x, y)$. However, if $Q[y/\bot] = \mathsf{B}(x)$ is satisfied by some tuple, then $[\![Q]\!]$ contains infinitely many tuples. Hence, $\mathsf{split}(Q) = ((\mathsf{B}(x) \vee \mathsf{P}(x, y)) \wedge \mathsf{P}(x, y), \exists x.\, \mathsf{B}(x))$. Because $Q_{fin} = (\mathsf{B}(x) \vee \mathsf{P}(x, y)) \wedge \mathsf{P}(x, y)$ is only used if $\not\models Q_{inf}$, i.e., if $\mathsf{B}(x)$ is never satisfied, we could simplify Q_{fin} to $\mathsf{P}(x, y)$. However, our translation does not implement such heuristic simplifications.

▶ **Example 11.** Consider the query $Q := \mathsf{B}(x) \wedge u \approx v$. The variables u and v are not range-restricted in Q and thus $\mathsf{split}(Q)$ chooses one of these variables (e.g., u) and restricts it by splitting Q into $Q_{\not\approx} = \mathsf{B}(x)$ and $Q^{=} = u \approx v$. Now, all variables are range-restricted in $Q_{\not\approx}$, but the variables in $Q_{\not\approx}$ and $Q^{=}$ are disjoint. Hence, $[\![Q]\!]$ contains infinitely many tuples whenever $Q_{\not\approx}$ is satisfied by some tuple. In contrast, $[\![Q]\!] = \emptyset$ if $Q_{\not\approx}$ is never satisfied. Hence, we have $\mathsf{split}(Q) = (\bot, \exists x.\, \mathsf{B}(x))$.

▶ **Example 12.** Consider the query $Q^{susp}_{user} := \mathsf{B}(b) \wedge \exists s.\, \forall p.\, \mathsf{P}(b, p) \longrightarrow \mathsf{S}(p, u, s)$ from Section 1. Restricting its bound variables yields the query $\mathsf{rb}(Q^{susp}_{user}) = \mathsf{B}(b) \wedge ((\exists s.\, (\neg\exists p.\, \mathsf{P}(b, p) \wedge \neg\mathsf{S}(p, u, s)) \wedge (\exists p.\, \mathsf{S}(p, u, s))) \vee (\neg\exists p.\, \mathsf{P}(b, p)))$ derived in Example 6. Splitting Q^{susp}_{user} yields

$$\mathsf{split}(Q^{susp}_{user}) = (\mathsf{rb}(Q^{susp}_{user}) \wedge (\exists s, p.\, \mathsf{S}(p, u, s)), \exists b.\, \mathsf{B}(b) \wedge \neg\exists p.\, \mathsf{P}(b, p)).$$

To understand $\mathsf{split}(Q^{susp}_{user})$, we apply (★) to $\mathsf{rb}(Q^{susp}_{user})$ for the free variable u:

$$\mathsf{rb}(Q^{susp}_{user}) \equiv (\mathsf{rb}(Q^{susp}_{user}) \wedge (\exists s, p.\, \mathsf{S}(p, u, s))) \vee (\mathsf{B}(b) \wedge (\neg\exists p.\, \mathsf{P}(b, p)) \wedge \neg\exists s, p.\, \mathsf{S}(p, u, s)).$$

If the subquery $\mathsf{B}(b) \wedge (\neg\exists p.\, \mathsf{P}(b,p))$ from the second disjunct is satisfied for some b, then Q^{susp}_{user} is satisfied by infinitely many values for u from the infinite domain $\mathcal{D}$ that do not belong to the finite interpretation of $\mathsf{S}(p,u,s)$ and thus satisfy the subquery $\neg\exists s,p.\, \mathsf{S}(p,u,s)$. Hence, $[\![Q^{susp}_{user}]\!]^{\mathcal{S}} = [\![\mathsf{rb}(Q^{susp}_{user})]\!]^{\mathcal{S}}$ is an infinite set of tuples whenever $\mathsf{B}(b) \wedge \neg\exists p.\, \mathsf{P}(b,p)$ is satisfied for some b. In contrast, if $\mathsf{B}(b) \wedge \neg\exists p.\, \mathsf{P}(b,p)$ is not satisfied for any b, then Q^{susp}_{user} is equivalent to $\mathsf{rb}(Q^{susp}_{user}) \wedge (\exists s,p.\, \mathsf{S}(p,u,s))$ obtained also by applying ($\bigstar$) to Q^{susp}_{user} for the free variable u.

▶ **Definition 13.** *Let Q be an RC query and $\mathsf{split}(Q) = (Q_{fin}, Q_{inf})$. Let $\hat{Q}_{fin} \coloneqq \mathsf{sr2ranf}(Q_{fin})$ and $\hat{Q}_{inf} \coloneqq \mathsf{sr2ranf}(Q_{inf})$ be the equivalent RANF queries. We define $\mathsf{rw}(Q) \coloneqq (\hat{Q}_{fin}, \hat{Q}_{inf})$.*

4.4 Complexity Analysis

In this section, we analyze the time complexity of capturing Q, i.e., checking if $[\![Q]\!]$ is finite and enumerating $[\![Q]\!]$ if it is finite. To bound the asymptotic time complexity of capturing a fixed Q, we ignore the (constant) time complexity of computing $\mathsf{rw}(Q) = (\hat{Q}_{fin}, \hat{Q}_{inf})$ and focus on the time complexity of evaluating the RANF queries $\hat{Q}_{fin}$ and $\hat{Q}_{inf}$, i.e., the query cost of $\hat{Q}_{fin}$ and $\hat{Q}_{inf}$. Without loss of generality, we assume that the input query Q has pairwise distinct (free and bound) variables to derive a set of quantified predicates from Q's atomic predicates and formulate our time complexity bound. Nevertheless, the RANF queries $\hat{Q}_{fin}$ and $\hat{Q}_{inf}$ computed by our translation need not have pairwise distinct (free and bound) variables.

Let $\mathsf{av}(Q)$ be the set of all (free and bound) variables in a query Q. We define the relation $\lesssim_Q$ on $\mathsf{av}(Q)$ such that $x \lesssim_Q y$ iff the scope of an occurrence of $x \in \mathsf{av}(Q)$ is contained in the scope of an occurrence of $y \in \mathsf{av}(Q)$. Formally, we define $x \lesssim_Q y$ iff $y \in \mathsf{fv}(Q)$ or $\exists x.\, Q_x \sqsubseteq \exists y.\, Q_y \sqsubseteq Q$ for some Q_x and Q_y. Note that $\lesssim_Q$ is a preorder on all variables and a partial order on the bound variables for every query with pairwise distinct (free and bound) variables.

Let $\mathsf{aps}(Q)$ be the set of all atomic predicates in a query Q. We denote by $\overline{\mathsf{qps}}(Q)$ the set of quantified predicates obtained from $\mathsf{aps}(Q)$ by performing the variable substitution $x \mapsto y$, where x and y are related by equalities in Q and $x \lesssim_Q y$, and existentially quantifying from a quantified predicate Q_{qp} the innermost bound variable x in Q that is free in Q_{qp}. Let $\mathsf{eqs}^*(Q)$ be the transitive closure of equalities occurring in Q. Formally, we define $\overline{\mathsf{qps}}(Q)$ by:

- $Q_{ap} \in \overline{\mathsf{qps}}(Q)$ if $Q_{ap} \in \mathsf{aps}(Q)$;
- $Q_{qp}[x \mapsto y] \in \overline{\mathsf{qps}}(Q)$ if $Q_{qp} \in \overline{\mathsf{qps}}(Q)$, $(x,y) \in \mathsf{eqs}^*(Q)$, and $x \lesssim_Q y$;
- $\exists x.\, Q_{qp} \in \overline{\mathsf{qps}}(Q)$ if $Q_{qp} \in \overline{\mathsf{qps}}(Q)$, $x \in \mathsf{fv}(Q_{qp}) \setminus \mathsf{fv}(Q)$, and $x \lesssim_Q y$ for all $y \in \mathsf{fv}(Q_{qp})$.

We bound the complexity of capturing Q by considering subsets $\mathcal{Q}_{qps}$ of quantified predicates $\overline{\mathsf{qps}}(Q)$ that are *minimal* in the sense that every quantified predicate in $\mathcal{Q}_{qps}$ contains a unique free variable that is not free in any other quantified predicate in $\mathcal{Q}_{qps}$. Formally, we define $\mathsf{minimal}(\mathcal{Q}_{qps}) \coloneqq \forall Q_{qp} \in \mathcal{Q}_{qps}.\, \mathsf{fv}(\mathcal{Q}_{qps} \setminus \{Q_{qp}\}) \neq \mathsf{fv}(\mathcal{Q}_{qps})$. Every minimal subset $\mathcal{Q}_{qps}$ of quantified predicates $\overline{\mathsf{qps}}(Q)$ contributes the product of the numbers of tuples satisfying each quantified predicate $Q_{qp} \in \mathcal{Q}_{qps}$ to the overall bound (that product is an upper bound on the number of tuples satisfying the join over all $Q_{qp} \in \mathcal{Q}_{qps}$). Similarly to Ngo et al. [22], we use the notation $\tilde{\mathcal{O}}(\cdot)$ to hide logarithmic factors incurred by set operations.

▶ **Theorem 14.** *Let Q be a fixed RC query with pairwise distinct (free and bound) variables. The time complexity of capturing Q, i.e., checking if $[\![Q]\!]$ is finite and enumerating $[\![Q]\!]$ if it is finite, is in $\tilde{\mathcal{O}}\left(\sum_{\mathcal{Q}_{qps} \subseteq \overline{\mathsf{qps}}(Q), \mathsf{minimal}(\mathcal{Q}_{qps})} \prod_{Q_{qp} \in \mathcal{Q}_{qps}} |[\![Q_{qp}]\!]|\right)$.*

We prove Theorem 14 in our extended report [25]. Examples 15 and 16 show that the time complexity from Theorem 14 cannot be achieved by the translation of Van Gelder and Topor [14] or over finite domains. Example 17 shows how equalities affect the bound in Theorem 14.

▶ **Example 15.** Consider the query $Q \coloneqq \mathsf{B}(b) \wedge \exists u, s.\, \neg\exists p.\, \mathsf{P}(b,p) \wedge \neg\mathsf{S}(p,u,s)$, equivalent to Q^{susp} from Section 1. Then $\mathsf{aps}(Q) = \{\mathsf{B}(b), \mathsf{P}(b,p), \mathsf{S}(p,u,s)\}$ and $\overline{\mathsf{qps}}(Q) = \{\mathsf{B}(b), \mathsf{P}(b,p), \exists p.\, \mathsf{P}(b,p), \mathsf{S}(p,u,s), \exists p.\, \mathsf{S}(p,u,s), \exists s, p.\, \mathsf{S}(p,u,s), \exists u, s, p.\, \mathsf{S}(p,u,s)\}$. The translated query Q_{vgt} by Van Gelder and Topor [14] restricts the variables r and s by $\exists s, p.\, \mathsf{S}(p,u,s)$ and $\exists u, p.\, \mathsf{S}(p,u,s)$, respectively. For an interpretation of B by $\{(\mathsf{c}') \mid \mathsf{c}' \in \{1,\ldots,n\}\}$, P by $\{(\mathsf{c}',\mathsf{c}') \mid \mathsf{c}' \in \{1,\ldots,n\}\}$, and S by $\{(\mathsf{c},\mathsf{c}',\mathsf{c}') \mid \mathsf{c} \in \{1,\ldots,n\}, \mathsf{c}' \in \{1,\ldots,m\}\}$, $n, m \in \mathbb{N}$, computing the join of $\mathsf{P}(b,p)$, $\exists s, p.\, \mathsf{S}(p,u,s)$, and $\exists u, p.\, \mathsf{S}(p,u,s)$, which is a Cartesian product, results in a time complexity in $\Omega(n \cdot m^2)$ for Q_{vgt}. In contrast, Theorem 14 yields an asymptotically better time complexity in $\tilde{\mathcal{O}}\,(n + m + n \cdot m)$ for our translation:

$$\tilde{\mathcal{O}}\,(|[\![\mathsf{B}(b)]\!]| + |[\![\mathsf{P}(b,p)]\!]| + |[\![\mathsf{S}(p,u,s)]\!]| + (|[\![\mathsf{B}(b)]\!]| + |[\![\mathsf{P}(b,p)]\!]|) \cdot |[\![\mathsf{S}(p,u,s)]\!]|)\,.$$

▶ **Example 16.** The query $\neg\mathsf{S}(x,y,z)$ is satisfied by a finite set of tuples over a finite domain $\mathcal{D}$ (as is every other query over a finite domain). For an interpretation of S by $\{(\mathsf{c},\mathsf{c},\mathsf{c}) \mid \mathsf{c} \in \mathcal{D}\}$, the equality $|\mathcal{D}| = |[\![\mathsf{S}(x,y,z)]\!]|$ holds and the number of satisfying tuples is

$$|[\![\neg\mathsf{S}(x,y,z)]\!]| = |\mathcal{D}|^3 - |[\![\mathsf{S}(x,y,z)]\!]| = |[\![\mathsf{S}(x,y,z)]\!]|^3 - |[\![\mathsf{S}(x,y,z)]\!]| \in \Omega(|[\![\mathsf{S}(x,y,z)]\!]|^3),$$

which exceeds the bound $\tilde{\mathcal{O}}\,(|[\![\mathsf{S}(x,y,z)]\!]|)$ of Theorem 14. Hence, our infinite domain assumption is crucial for achieving the better complexity bound.

▶ **Example 17.** Consider the following query over the domain $\mathcal{D} = \mathbb{N}$ of natural numbers:

$$Q \coloneqq \forall u.\, (u \approx 0 \vee u \approx 1 \vee u \approx 2) \longrightarrow \\ (\exists v.\, \mathsf{B}(v) \wedge (u \approx 0 \longrightarrow x \approx v) \wedge (u \approx 1 \longrightarrow y \approx v) \wedge (u \approx 2 \longrightarrow z \approx v)).$$

Note that this query is equivalent to $Q \equiv \mathsf{B}(x) \wedge \mathsf{B}(y) \wedge \mathsf{B}(z)$ and thus it is satisfied by a finite set of tuples of size $|[\![\mathsf{B}(x)]\!]| \cdot |[\![\mathsf{B}(y)]\!]| \cdot |[\![\mathsf{B}(z)]\!]| = |[\![\mathsf{B}(x)]\!]|^3$. The set of atomic predicates of Q is $\mathsf{aps}(Q) = \{\mathsf{B}(v)\}$ and it must be closed under the equalities occurring in Q to yield a valid bound in Theorem 14. In this case, $\overline{\mathsf{qps}}(Q) = \{\mathsf{B}(v), \exists v.\, \mathsf{B}(v), \mathsf{B}(x), \mathsf{B}(y), \mathsf{B}(z)\}$ and the bound in Theorem 14 is $|[\![\mathsf{B}(v)]\!]| \cdot |[\![\mathsf{B}(x)]\!]| \cdot |[\![\mathsf{B}(y)]\!]| \cdot |[\![\mathsf{B}(z)]\!]| = |[\![\mathsf{B}(x)]\!]|^4$. In particular, this bound is not tight, but it still reflects the complexity of evaluating the RANF queries produced by our translation as it does not derive the equivalence $Q \equiv \mathsf{B}(x) \wedge \mathsf{B}(y) \wedge \mathsf{B}(z)$.

5 Data Golf Benchmark

In this section, we devise the *Data Golf* benchmark for generating structures for given RC queries. We will use the benchmark in our empirical evaluation (Section 6). Given an RC query, we seek a structure that results in a nontrivial evaluation result for the overall query and for all its subqueries. Intuitively, the resulting structure makes query evaluation potentially more challenging compared to the case where some subquery results in a trivial (e.g., empty) evaluation result. More specifically, Data Golf has two objectives. The first resembles the *regex golf* game's objective [11] (hence the name) and aims to find a structure on which the result of a given query contains a given *positive* set of tuples and does not contain any tuples from another given *negative* set. The second objective is to ensure that all the query's subqueries evaluate to a non-trivial result.

Algorithm 3 Computing the Data Golf structure.

input: An RC query Q with pairwise distinct (free and bound) variables satisfying CON, CST, VAR, REP, a sequence of distinct variables $\vec{v}$, $\mathsf{fv}(Q) \subseteq \vec{v}$, sets of tuples $\mathcal{T}^+_{\vec{v}}$ and $\mathcal{T}^-_{\vec{v}}$ over $\vec{v}$ such that $\left|\mathcal{T}^+_{\vec{v}}[x]\right| = \left|\mathcal{T}^+_{\vec{v}}\right|$, $\left|\mathcal{T}^-_{\vec{v}}[x]\right| = \left|\mathcal{T}^-_{\vec{v}}\right|$, and $\mathcal{T}^+_{\vec{v}}[x] \cap \mathcal{T}^-_{\vec{v}}[x] = \emptyset$, for every $x \in \vec{v}$, a parameter $\gamma \in \{0, 1\}$.

output: A structure $\mathcal{S}$ such that $\mathcal{T}^+_{\vec{v}}[\vec{\mathsf{fv}}(Q)] \subseteq [\![Q]\!]$, $\mathcal{T}^-_{\vec{v}}[\vec{\mathsf{fv}}(Q)] \cap [\![Q]\!] = \emptyset$, and $|[\![Q']\!]|$ and $|[\![\neg Q']\!]|$ contain at least $\min\{\left|\mathcal{T}^+_{\vec{v}}\right|, \left|\mathcal{T}^-_{\vec{v}}\right|\}$ tuples, for every $Q' \sqsubseteq Q$.

function $\mathsf{dg}(Q, \vec{v}, \mathcal{T}^+_{\vec{v}}, \mathcal{T}^-_{\vec{v}}, \gamma) =$
  switch Q **do**
    case $r(t_1, \ldots, t_{\iota(r)})$ **do return** $\{r^{\mathcal{S}} \mapsto \mathcal{T}^+_{\vec{v}}[t_1, \ldots, t_{\iota(r)}]\}$;
    case $x \approx y$ **do**
      if *there exist* d, d' *such that* $d \neq d'$ *and* $(d, d') \in \mathcal{T}^+_{\vec{v}}[x, y]$, *or* $d = d'$ *and* $(d, d') \in \mathcal{T}^-_{\vec{v}}[x, y]$ **then** fail;
    case $\neg Q'$ **do return** $\mathsf{dg}(Q', \vec{v}, \mathcal{T}^-_{\vec{v}}, \mathcal{T}^+_{\vec{v}}, \gamma)$;
    case $Q_1 \vee Q_2$ *or* $Q_1 \wedge Q_2$ **do**
      $(\mathcal{T}^1_{\vec{v}}, \mathcal{T}^2_{\vec{v}}) \leftarrow \{(\mathcal{T}^1_{\vec{v}}, \mathcal{T}^2_{\vec{v}}) \mid \left|\mathcal{T}^1_{\vec{v}}[x]\right| = \left|\mathcal{T}^2_{\vec{v}}[x]\right| = \left|\mathcal{T}^1_{\vec{v}}\right| = \left|\mathcal{T}^2_{\vec{v}}\right| = \min\{\left|\mathcal{T}^+_{\vec{v}}\right|, \left|\mathcal{T}^-_{\vec{v}}\right|\}$, $\mathcal{T}^1_{\vec{v}}[x] \cap \mathcal{T}^2_{\vec{v}}[x] = \emptyset, (\mathcal{T}^1_{\vec{v}}[x] \cup \mathcal{T}^2_{\vec{v}}[x]) \cap (\mathcal{T}^+_{\vec{v}}[x] \cup \mathcal{T}^-_{\vec{v}}[x]) = \emptyset$, for all $x \in \vec{v}\}$;
      if $\gamma = 0$ **then**
        return $\mathsf{dg}(Q_1, \vec{v}, \mathcal{T}^+_{\vec{v}} \cup \mathcal{T}^1_{\vec{v}}, \mathcal{T}^-_{\vec{v}} \cup \mathcal{T}^2_{\vec{v}}, \gamma) \cup \mathsf{dg}(Q_2, \vec{v}, \mathcal{T}^+_{\vec{v}} \cup \mathcal{T}^2_{\vec{v}}, \mathcal{T}^-_{\vec{v}} \cup \mathcal{T}^1_{\vec{v}}, \gamma)$;
      else
        switch Q **do**
          case $Q_1 \vee Q_2$ **do**
            return $\mathsf{dg}(Q_1, \vec{v}, \mathcal{T}^+_{\vec{v}} \cup \mathcal{T}^1_{\vec{v}}, \mathcal{T}^-_{\vec{v}} \cup \mathcal{T}^2_{\vec{v}}, \gamma) \cup \mathsf{dg}(Q_2, \vec{v}, \mathcal{T}^1_{\vec{v}} \cup \mathcal{T}^2_{\vec{v}}, \mathcal{T}^-_{\vec{v}} \cup \mathcal{T}^+_{\vec{v}}, \gamma)$;
          case $Q_1 \wedge Q_2$ **do**
            return $\mathsf{dg}(Q_1, \vec{v}, \mathcal{T}^+_{\vec{v}} \cup \mathcal{T}^-_{\vec{v}}, \mathcal{T}^1_{\vec{v}} \cup \mathcal{T}^2_{\vec{v}}, \gamma) \cup \mathsf{dg}(Q_2, \vec{v}, \mathcal{T}^+_{\vec{v}} \cup \mathcal{T}^2_{\vec{v}}, \mathcal{T}^-_{\vec{v}} \cup \mathcal{T}^1_{\vec{v}}, \gamma)$;
    case $\exists y.\, Q_y$ **do**
      $(\mathcal{T}^1_{\vec{v}\cdot y}, \mathcal{T}^2_{\vec{v}\cdot y}) \leftarrow \{(\mathcal{T}^1_{\vec{v}\cdot y}, \mathcal{T}^2_{\vec{v}\cdot y}) \mid \mathcal{T}^1_{\vec{v}\cdot y}[\vec{v}] = \mathcal{T}^+_{\vec{v}}, \mathcal{T}^2_{\vec{v}\cdot y}[\vec{v}] = \mathcal{T}^-_{\vec{v}}$, $\left|\mathcal{T}^1_{\vec{v}\cdot y}[y]\right| = \left|\mathcal{T}^1_{\vec{v}\cdot y}\right| = \left|\mathcal{T}^+_{\vec{v}}\right|, \left|\mathcal{T}^2_{\vec{v}\cdot y}[y]\right| = \left|\mathcal{T}^2_{\vec{v}\cdot y}\right| = \left|\mathcal{T}^-_{\vec{v}}\right|, \mathcal{T}^1_{\vec{v}\cdot y}[y] \cap \mathcal{T}^2_{\vec{v}\cdot y}[y] = \emptyset\}$;
      return $\mathsf{dg}(Q_y, \vec{v} \cdot y, \mathcal{T}^1_{\vec{v}\cdot y}, \mathcal{T}^2_{\vec{v}\cdot y}, \gamma)$;

Formally, given a query Q and two sets of tuples $\mathcal{T}^+$ and $\mathcal{T}^-$ over a fixed domain $\mathcal{D}$, representing assignments of $\mathsf{fv}(Q)$, Data Golf produces a structure $\mathcal{S}$ (represented as a partial mapping from predicate symbols to their interpretations), such that $\mathcal{T}^+ \subseteq [\![Q]\!]$, $\mathcal{T}^- \cap [\![Q]\!] = \emptyset$, and $|[\![Q']\!]|$ and $|[\![\neg Q']\!]|$ contain at least $\min\{|\mathcal{T}^+|, |\mathcal{T}^-|\}$ tuples, for every $Q' \sqsubseteq Q$. To be able to produce such a structure $\mathcal{S}$, we make the following assumptions on Q:

CON the bound variable y in every subquery $\exists y.\, Q_y$ of Q satisfies $\mathsf{con}(y, Q_y, \mathcal{G})$ [14, Figure 5] for some set $\mathcal{G}$ such that $\mathsf{eqs}(y, \mathcal{G}) = \emptyset$ and, for every $Q_{qp} \in \mathcal{G}$, $\{y\} \subsetneq \mathsf{fv}(Q_{qp})$ holds; this avoids subqueries like $\exists y.\, \neg \mathsf{P}_2(x, y)$ and $\exists y.\, (\mathsf{P}_2(x, y) \vee \mathsf{P}_1(y))$;

CST Q contains no subquery of the form $x \approx \mathsf{c}$, which is satisfied by exactly one tuple;

VAR Q contains no closed subqueries, e.g., $\mathsf{P}_1(42)$, because a closed subquery is either satisfied by all possible tuples or no tuple at all; and

REP Q contains no repeated predicate symbols; this avoids subqueries like $\mathsf{P}_1(x) \wedge \neg \mathsf{P}_1(x)$.

Given a sequence of pairwise distinct variables $\vec{v}$ and a tuple $\vec{d}$ of the same length, we may interpret the tuple $\vec{d}$ as a *tuple over* $\vec{v}$, denoted as $\vec{d}(\vec{v})$. Given a sequence $t_1, \ldots, t_k \in \vec{v} \cup \mathcal{C}$ of terms, we denote by $\vec{d}(\vec{v})[t_1, \ldots, t_k]$ the tuple obtained by evaluating the terms $t_1, \ldots, t_k$ over $\vec{d}(\vec{v})$. Formally, we define $\vec{d}(\vec{v})[t_1, \ldots, t_k] \coloneqq (d'_i)_{i=1}^k$, where $d'_i = \vec{d}_j$ if $t_i = \vec{v}_j$ and $d'_i = t_i$ if $t_i \in \mathcal{C}$. We lift this notion to sets of tuples over $\vec{v}$ in the standard way.

Data Golf is formalized by the function $\mathsf{dg}(Q, \vec{v}, \mathcal{T}^+_{\vec{v}}, \mathcal{T}^-_{\vec{v}}, \gamma)$, defined in Algorithm 3, where $\vec{v}$ is a sequence of distinct variables such that $\mathsf{fv}(Q) \subseteq \vec{v}$, $\mathcal{T}^+_{\vec{v}}$ and $\mathcal{T}^-_{\vec{v}}$ are sets of tuples over $\vec{v}$, and $\gamma \in \{0, 1\}$ is a *strategy*. The function $\mathsf{dg}(Q, \vec{v}, \mathcal{T}^+_{\vec{v}}, \mathcal{T}^-_{\vec{v}}, \gamma)$ can fail on an equality between two variables $x \approx y$. In this case, the function $\mathsf{dg}(Q, \vec{v}, \mathcal{T}^+_{\vec{v}}, \mathcal{T}^-_{\vec{v}}, \gamma)$ does not compute a Data Golf structure. We define the *not-depth* of a subquery $x \approx y$ in Q as the number of subqueries that have the form of a negation among the queries $x \approx y \sqsubseteq \cdots \sqsubseteq Q$, i.e., the number of negations on the path between the subquery $x \approx y$ and Q's main connective. To prevent failure, we generate the sets $\mathcal{T}^+_{\vec{v}}$, $\mathcal{T}^-_{\vec{v}}$ to only contain tuples with equal values for all variables in equalities with even (odd, respectively) not-depth and pairwise distinct values for all variables in equalities with odd (even, respectively) not-depth. This is not always possible, e.g., for $x \approx y \wedge \neg x \approx y$, in which case no Data Golf structure can be computed. In the case of a conjunction or a disjunction, we add disjoint sets $\mathcal{T}^1_{\vec{v}}$, $\mathcal{T}^2_{\vec{v}}$ of tuples over $\vec{v}$ to $\mathcal{T}^+_{\vec{v}}$, $\mathcal{T}^-_{\vec{v}}$ so that the intermediate results for the subqueries are neither equal nor disjoint. We implement two strategies (parameter γ) to choose these sets $\mathcal{T}^1_{\vec{v}}$, $\mathcal{T}^2_{\vec{v}}$.

Finally, we justify why a Data Golf structure $\mathcal{S}$ computed by $\mathsf{dg}(Q, \vec{v}, \mathcal{T}^+_{\vec{v}}, \mathcal{T}^-_{\vec{v}}, \gamma)$ satisfies $\mathcal{T}^+_{\vec{v}}[\vec{\mathsf{fv}}(Q)] \subseteq [\![Q]\!]$ and $\mathcal{T}^-_{\vec{v}}[\vec{\mathsf{fv}}(Q)] \cap [\![Q]\!] = \emptyset$. We proceed by induction on the query Q. Because of REP, the Data Golf structures for the subqueries Q_1, Q_2 of a binary query $Q_1 \vee Q_2$ or $Q_1 \wedge Q_2$ can be combined using the union operator. The only case that does not follow immediately is that $\mathcal{T}^-_{\vec{v}}[\vec{\mathsf{fv}}(Q)] \cap [\![Q]\!] = \emptyset$ for a query Q of the form $\exists y.\, Q_y$. We prove this case by contradiction. Without loss of generality we assume that $\vec{\mathsf{fv}}(Q_y) = \vec{\mathsf{fv}}(Q) \cdot y$. Suppose that $\vec{d} \in \mathcal{T}^-_{\vec{v}}[\vec{\mathsf{fv}}(Q)]$ and $\vec{d} \in [\![Q]\!]$. Because $\vec{d} \in \mathcal{T}^-_{\vec{v}}[\vec{\mathsf{fv}}(Q)]$, there exists some d such that $\vec{d} \cdot d \in \mathcal{T}^2_{\vec{v}\cdot y}[\vec{\mathsf{fv}}(Q_y)]$. Because $\vec{d} \in [\![Q]\!]$, there exists some d' such that $\vec{d} \cdot d' \in [\![Q_y]\!]$. By the induction hypothesis, $\vec{d} \cdot d \notin [\![Q_y]\!]$ and $\vec{d} \cdot d' \notin \mathcal{T}^2_{\vec{v}\cdot y}[\vec{\mathsf{fv}}(Q_y)]$. Because $\mathsf{con}(y, Q_y, \mathcal{G})$ holds for some $\mathcal{G}$ satisfying CON, the query Q_y is equivalent to $(Q_y \wedge \mathsf{qps}^\vee(\mathcal{G})) \vee Q_y[y/\bot]$. We have $\vec{d} \cdot d' \in [\![Q_y]\!]$. If the tuple $\vec{d} \cdot d'$ satisfies $Q_y[y/\bot]$, then $\vec{d} \cdot d \in [\![Q_y]\!]$ (contradiction) because the variable y does not occur in the query $Q_y[y/\bot]$ and thus its assignment in $\vec{d} \cdot d'$ can be arbitrarily changed. Otherwise, the tuple $\vec{d} \cdot d'$ satisfies some quantified predicate $Q_{qp} \in \mathsf{qps}(\mathcal{G})$ and (CON) implies $\{y\} \subsetneq \mathsf{fv}(Q_{qp})$. Hence, the tuples $\vec{d} \cdot d$ and $\vec{d} \cdot d'$ agree on the assignment of a variable $x \in \mathsf{fv}(Q_{qp}) \setminus \{y\}$. Let $\mathcal{T}^+_{\vec{v}'}$ and $\mathcal{T}^-_{\vec{v}'}$ be the sets in the recursive call of dg on the atomic predicate from Q_{qp}. Because $\vec{d} \cdot d \in \mathcal{T}^2_{\vec{v}\cdot y}[\vec{\mathsf{fv}}(Q_y)]$ and $\mathcal{T}^2_{\vec{v}\cdot y}[\vec{\mathsf{fv}}(Q_y)] \subseteq \mathcal{T}^+_{\vec{v}'}[\vec{\mathsf{fv}}(Q_y)] \cup \mathcal{T}^-_{\vec{v}'}[\vec{\mathsf{fv}}(Q_y)]$, the tuple $\vec{d} \cdot d$ is in $\mathcal{T}^+_{\vec{v}'}[\vec{\mathsf{fv}}(Q_y)] \cup \mathcal{T}^-_{\vec{v}'}[\vec{\mathsf{fv}}(Q_y)]$. Because $\vec{d} \cdot d'$ satisfies the quantified predicate Q_{qp}, the tuple $\vec{d} \cdot d'$ is in $\mathcal{T}^+_{\vec{v}'}[\vec{\mathsf{fv}}(Q_y)]$. Next we observe that the assignments of every variable (in particular, x) in the tuples from the sets $\mathcal{T}^+_{\vec{v}'}$, $\mathcal{T}^-_{\vec{v}'}$ are pairwise distinct (the conditions $\mathcal{T}^+_{\vec{v}'}[x] \cap \mathcal{T}^-_{\vec{v}'}[x] = \emptyset$, $\left|\mathcal{T}^+_{\vec{v}'}[x]\right| = \left|\mathcal{T}^+_{\vec{v}'}\right|$, and $\left|\mathcal{T}^-_{\vec{v}'}[x]\right| = \left|\mathcal{T}^-_{\vec{v}'}\right|$). Because the tuples $\vec{d} \cdot d$ and $\vec{d} \cdot d'$ agree on the assignment of x, they must be equal, i.e., $\vec{d} \cdot d = \vec{d} \cdot d'$ (contradiction).

The sets $\mathcal{T}^+_{\vec{v}}$, $\mathcal{T}^-_{\vec{v}}$ only grow in dg's recursion and the properties CON, CST, VAR, REP imply that Q has no closed subquery. Hence, $\mathcal{T}^+_{\vec{v}}[\vec{\mathsf{fv}}(Q)] \subseteq [\![Q]\!]$ and $\mathcal{T}^-_{\vec{v}}[\vec{\mathsf{fv}}(Q)] \cap [\![Q]\!] = \emptyset$ imply that $|[\![Q']\!]|$ and $|[\![\neg Q']\!]|$ contain at least $\min\{\left|\mathcal{T}^+_{\vec{v}}\right|, \left|\mathcal{T}^-_{\vec{v}}\right|\}$ tuples, for every $Q' \sqsubseteq Q$.

▶ **Example 18.** Consider the query $Q \coloneqq \neg\exists y.\, \mathsf{P}_2(x, y) \wedge \neg\mathsf{P}_3(x, y, z)$. This query Q satisfies the assumptions CON, CST, VAR, REP. In particular, $\mathsf{con}(y, \mathsf{P}_2(x, y) \wedge \neg\mathsf{P}_3(x, y, z), \mathcal{G})$ holds for $\mathcal{G} = \{\mathsf{P}_2(x, y)\}$ with $\{y\} \subsetneq \mathsf{fv}(\mathsf{P}_2(x, y))$. We choose $\vec{v} = (x, z)$, $\mathcal{T}^+_{\vec{v}} = \{(0, 4), (2, 6)\}$, and $\mathcal{T}^-_{\vec{v}} = \{(8, 12), (10, 14)\}$. The function $\mathsf{dg}(Q, \vec{v}, \mathcal{T}^+_{\vec{v}}, \mathcal{T}^-_{\vec{v}}, \gamma)$ first flips $\mathcal{T}^+_{\vec{v}}$ and $\mathcal{T}^-_{\vec{v}}$ (because Q's main connective is negation) and then extends the tuples in the sets $\mathcal{T}^-_{\vec{v}}$ and $\mathcal{T}^+_{\vec{v}}$ with a value for the bound variable y: $\mathcal{T}^1_{\vec{v}\cdot y} = \{(8, 12, 16), (10, 14, 18)\}$ and $\mathcal{T}^2_{\vec{v}\cdot y} = \{(0, 4, 20), (2, 6, 22)\}$.

For conjunction (a binary operator), two additional sets of tuples are computed: $\overline{\mathcal{T}^1_{\vec{v}\cdot y}} = \{(24, 28, 32), (26, 30, 34)\}$ and $\overline{\mathcal{T}^2_{\vec{v}\cdot y}} = \{(36, 40, 44), (38, 42, 46)\}$. Depending on the strategy ($\gamma = 0$ or $\gamma = 1$), one of the following structures is computed: $\mathcal{S}_0 = \{\mathsf{P}_2 \mapsto \{(8, 16), (10, 18), (24, 32), (26, 34)\}, \mathsf{P}_3 \mapsto \mathcal{T}^+_{xyz}\}$, or $\mathcal{S}_1 = \{\mathsf{P}_2 \mapsto \{(8, 16), (10, 18), (0, 20), (2, 22)\}, \mathsf{P}_3 \mapsto \mathcal{T}^+_{xyz}\}$, where $\mathcal{T}^+_{xyz} = \{(0, 20, 4), (2, 22, 6), (24, 32, 28), (26, 34, 30)\}$.

The query $\mathsf{P}_1(x) \wedge Q$ is satisfied by the finite set of tuples $\mathcal{T}^+_{\vec{v}}$ under the structure $\mathcal{S}_1 \cup \{\mathsf{P}_1 \mapsto \{(0),(2)\}\}$ obtained by extending $\mathcal{S}_1$ ($\gamma = 1$). In contrast, the same query $\mathsf{P}_1(x) \wedge Q$ is satisfied by an infinite set of tuples including $\mathcal{T}^+_{\vec{v}}$ and disjoint from $\mathcal{T}^-_{\vec{v}}$ under the structure $\mathcal{S}_0 \cup \{\mathsf{P}_1 \mapsto \{(0),(2)\}\}$ obtained by extending $\mathcal{S}_0$ ($\gamma = 0$).

6 Implementation and Empirical Evaluation

We have implemented our translation RC2SQL consisting of roughly 1000 lines of OCaml code [25]. Although our translation satisfies the worst-case complexity bound (Theorem 14), we further improve its average-case complexity by implementing the following optimizations, described in more detail in our extended report [25, Section E].

- We use a sample structure of constant size, called a *training database*, to estimate the query cost when resolving the nondeterministic choices in our algorithms. A good training database should preserve the relative ordering of queries by their cost over the actual database as much as possible. Nevertheless, our translation satisfies the correctness and worst-case complexity claims (Section 4.3 and 4.4) for every choice of the training database. All our experiments used a Data Golf structure with $|\mathcal{T}^+| = |\mathcal{T}^-| = 2$ as the training database.
- We use the function optcnt optimizing RANF subqueries of the form $\exists \vec{y}.\, Q^+ \wedge \bigwedge_{i=1}^{k} \neg Q_i^-$ using the count aggregation operator. Inspired by Claußen et al. [9], we compare the number of assignments of $\vec{y}$ that satisfy Q^+ and $\bigvee_{i=1}^{k}(Q^+ \wedge Q_i^-)$, respectively.
- To compute an SQL query from a RANF query, we define the function ranf2sql$(\cdot)$. We first obtain an equivalent RA expression using the standard approach [1] but adjusting the case of closed queries [8]. To translate RA expressions into SQL, we reuse a publicly available RA interpreter radb [30]. We modify its implementation to improve the performance of the resulting SQL query. We map the anti-join operator $\hat{Q}_1 \triangleright \hat{Q}_2$ to a more efficient LEFT JOIN, if $\mathsf{fv}(\hat{Q}_2) \subsetneq \mathsf{fv}(\hat{Q}_1)$, and we perform common subquery elimination.

To validate our translation's improved asymptotic time complexity, we compare it with the translation by Van Gelder and Topor [14] (VGT), an implementation of the algorithm by Ailamazyan et al. [2] that uses an extended active domain as the generators, and the DDD [20, 21], LDD [7], and MonPoly$^{\mathsf{REG}}$ [4] tools that support direct RC query evaluation using binary decision diagrams. We could not find a publicly available implementation of Van Gelder and Topor's translation. Therefore, the tool VGT for evaluable RC queries is derived from our implementation by modifying the function rb$(\cdot)$ in Algorithm 1 to use the con relation [14, Figure 5] instead of cov$(x, Q, \mathcal{G})$ (Figure 3) and to use the generator $\exists \vec{\mathsf{fv}}(Q) \setminus \{x\}.\, \mathsf{qps}^{\vee}(\mathcal{G})$ instead of $\mathsf{qps}^{\vee}(\mathcal{G})$. Evaluable queries Q are always translated into $(Q_{fin}, \bot)$ by rw$(\cdot)$ because all of Q's free variables are range-restricted. We also consider translation variants that omit the count aggregation optimization optcnt$(\cdot)$, marked with a minus $(^-)$.

SQL queries computed by the translations are evaluated using the PostgreSQL database engine. We have also used the MySQL database engine but omit its timings from our evaluation after discovering that it computed incorrect results for some queries. This issue was reported and subsequently confirmed by MySQL developers. We run our experiments on an Intel Core i5-4200U CPU computer with 8 GB RAM. The relations in PostgreSQL are recreated before each invocation to prevent optimizations based on caching recent query evaluation results. We provide all our experiments in an easily reproducible artifact [25].

In the Small, Medium, and Large experiments, we generate ten pseudorandom queries with a fixed size 14 and Data Golf structures $\mathcal{S}$. The queries satisfy the Data Golf assumptions along with a few additional ones: the queries are not safe-range, have no repeated equalities,

Experiment Small, Evaluable pseudorandom queries Q, $|\mathsf{sub}(Q)| = 14$, $n = 500$:

RC2SQL	**0.3**	0.3	**0.3**	**0.2**	**0.2**	**0.3**	**0.3**	**0.2**	**0.2**	0.3
RC2SQL$^-$	**0.3**	**0.2**	150.3	0.3	**0.2**	**0.3**	**0.3**	0.3	5.9	**0.2**
VGT	31.5	6.7	4.2	2.5	37.5	9.3	2.4	2.3	11.3	2.7
VGT$^-$	33.7	4.8	119.9	6.3	11.2	21.9	31.4	11.3	12.3	21.9
DDD	9.1	2.5	RE	7.1	5.9	RE	5.1	RE	2.2	5.1
LDD	59.2	24.1	169.1	38.8	53.3	37.4	64.0	TO	16.0	61.6
MonPoly$^{\mathsf{REG}}$	64.2	31.4	143.0	57.6	67.8	54.4	72.4	174.6	33.6	71.3

Experiment Medium, Evaluable pseudorandom queries Q, $|\mathsf{sub}(Q)| = 14$, $n = 20000$:

RC2SQL	2.6	1.4	**3.9**	2.1	**1.5**	2.8	3.3	**1.6**	**1.2**	2.6
RC2SQL$^-$	**2.0**	**1.0**	TO	**2.0**	1.7	**2.5**	**2.3**	1.8	TO	**1.8**
VGT	TO	TO	7.8	3.9	TO	TO	5.2	4.7	TO	4.8
VGT$^-$	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO

Experiment Large, Evaluable pseudorandom queries Q, $|\mathsf{sub}(Q)| = 14$, tool = RC2SQL:

$n = 40000$	3.5	2.7	8.1	4.0	3.2	5.5	6.7	4.1	1.9	5.8
$n = 80000$	7.5	5.4	16.1	8.0	6.1	11.5	14.0	8.1	4.2	11.7
$n = 120000$	13.2	8.2	24.6	11.5	8.9	16.3	20.9	11.0	7.2	16.7

Experiment Infinite, Non-evaluable pseudorandom queries Q, $|\mathsf{sub}(Q)| = 7$, $n = 4000$:

	Infinite results ($\gamma = 0$)					Finite results ($\gamma = 1$)				
RC2SQL	0.8	0.8	0.8	0.8	0.8	1.0	1.1	0.9	**2.4**	**1.1**
RC2SQL$^-$	**0.5**	**0.5**	**0.4**	**0.5**	**0.5**	**0.6**	**0.7**	**0.6**	TO	2.0
DDD	89.5	49.1	46.9	116.3	50.4	81.7	44.1	45.8	89.8	44.6
LDD	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO
MonPoly$^{\mathsf{REG}}$	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO

Figure 4 Experiments Small, Medium, Large, and Infinite. We use the following abbreviations: TO = Timeout of 300s, RE = Runtime Error.

disjunction only appears at the top-level, every bound variable actually occurs in its scope, and only pairwise distinct variables appear as terms in predicates. The queries have 2 free variables and every subquery has at most 4 free variables. We control the size of the Data Golf structure $\mathcal{S}$ in our experiments using a parameter $n = |\mathcal{T}^+| = |\mathcal{T}^-|$. Because the sets $\mathcal{T}^+$ and $\mathcal{T}^-$ grow in the recursion on subqueries, relations in a Data Golf structure typically have more than n tuples. The values of the parameter n for Data Golf structures are summarized in Figure 4.

The Infinite experiment consists of five pseudorandom queries Q that are *not* evaluable and $\mathsf{rw}(Q) = (Q_{fin}, Q_{inf})$, where $Q_{inf} \neq \bot$. Specifically, the queries are of the form $Q_1 \wedge \forall x, y.\, Q_2 \longrightarrow Q_3$, where Q_1, Q_2, and Q_3 are either atomic predicates or equalities. For each query Q, we compare the performance of our tool to tools that directly evaluate Q on structures generated by the two Data Golf strategies (parameter γ), which trigger infinite or finite evaluation results on the considered queries. For infinite results, our tool outputs this fact (by evaluating Q_{inf}), whereas the other tools also output a finite representation of the infinite result. For finite results, all tools produce the same output.

Figure 4 shows the empirical evaluation results for the experiments Small, Medium, Large, and Infinite. All entries are execution times in seconds, TO is a timeout, and RE is a runtime error. Each column shows evaluation times for a unique pseudorandom query. The lowest time for a query is typeset in bold. We do not report the translation time because it does not contribute to the time complexity for a fixed query. Still, RC2SQL's translation time is at most 0.6 seconds on every query in our experiments. We also omit the rows for

Query	Q^{susp}		Q^{susp}_{user}		Q^{susp}_{text}	
Param. n	10^3	10^4	10^3	10^4	10^3	10^4
RC2SQL	**2.0**	**2.2**	**3.0**	**3.5**	**6.2**	**7.1**
RC2SQL$^-$	61.7	TO	63.4	TO	484.9	TO
VGT	3.9	2.9	–	–	213.2	TO
VGT$^-$	433.8	TO	–	–	495.4	TO
DDD	7.1	TO	6.3	TO	28.8	TO
LDD	36.3	TO	34.0	TO	213.9	TO
MonPoly$^{\text{REG}}$	49.9	TO	47.3	TO	181.2	TO

Query	Q^{susp}		Q^{susp}_{user}		Q^{susp}_{text}	
Dataset	GC	MI	GC	MI	GC	MI
RC2SQL	**2.9**	**16.2**	**4.2**	**21.4**	**8.9**	**91.3**
RC2SQL$^-$	273.9	TO	270.1	TO	TO	TO
VGT	3.5	18.9	–	–	TO	TO
VGT$^-$	TO	TO	–	–	TO	TO
DDD	93.3	TO	90.1	TO	178.5	TO
LDD	TO	TO	TO	TO	TO	TO
MonPoly$^{\text{REG}}$	TO	TO	TO	TO	TO	TO

Figure 5 Experiment with the queries Q^{susp}, Q^{susp}_{user}, and Q^{susp}_{text}. We use the following abbreviations: GC = Gift Cards dataset, MI = Musical Instruments dataset, TO = Timeout of 600s.

tools that time out or crash on all queries of an experiment, e.g., Ailamazyan et al. [2]. We conclude that our translation RC2SQL significantly outperforms all other tools on all queries and scales well to higher values of n, i.e., larger relations in the Data Golf structures, on all queries.

We also evaluate the tools on the queries Q^{susp} and Q^{susp}_{user} from the introduction and on the more challenging query $Q^{susp}_{text} \coloneqq \mathsf{B}(b) \wedge \exists u, s, t.\ \forall p.\ \mathsf{P}(b, p) \longrightarrow \mathsf{S}(p, u, s) \vee \mathsf{T}(p, u, t)$ with an additional relation T that relates user's review text (variable t) to a product. The query Q^{susp}_{text} computes all brands for which there is a user, a score, and a review text such that all the brand's products were reviewed by that user with that score or by that user with that text. We use both Data Golf structures (strategy $\gamma = 1$) and real-world structures obtained from the Amazon review dataset [23]. The real-world relations P, S, and T are obtained by projecting the respective tables from the Amazon review dataset for some chosen product categories (abbreviated GC and MI in Figure 5) and the relation B contains all brands from P that have at least three products. Because the tool by Ailamazyan et al., DDD, LDD, and MonPoly$^{\text{REG}}$ only support integer data, we injectively remap the string and floating-point values from the Amazon review dataset to integers.

Figure 5 shows the empirical evaluation results: execution times on Data Golf structures (left) and execution times on structures derived from the real-world dataset for two specific product categories (right). We remark that VGT cannot handle the query Q^{susp}_{user} as it is not evaluable [14]. Our translation RC2SQL significantly outperforms all other tools (except VGT on Q^{susp}, but RC2SQL still outperforms VGT) on both Data Golf and real-world structures. VGT$^-$ translates Q^{susp} into a RANF query with a higher query cost than RC2SQL$^-$. However, the optimization optcnt$(\cdot)$ manages to rectify this inefficiency and thus VGT exhibits a comparable performance as RC2SQL. Specifically, the factor of 80× in query cost between VGT$^-$ and RC2SQL$^-$ improves to 1.1× in query cost between VGT and RC2SQL on a Data Golf structure with $n = 20$ [25]. Nevertheless, VGT does not finish evaluating the query Q^{susp}_{text} on GC and MI datasets within 10 minutes, unlike RC2SQL.

7 Conclusion

We presented a translation-based approach to evaluating arbitrary relational calculus queries over an infinite domain with improved time complexity over existing approaches. This contribution is an important milestone towards making the relational calculus a viable query language for practical databases. In future work, we plan to integrate into our base language features that database practitioners love, such as inequalities, bag semantics, or aggregations.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

2 Alfred K. Ailamazyan, Mikhail M. Gilula, Alexei P. Stolboushkin, and Grigorii F. Schwartz. Reduction of a relational model with infinite domains to the case of finite domains. *Doklady Akademii Nauk SSSR*, 286(2):308–311, 1986. URL: http://mi.mathnet.ru/dan47310.

3 Arnon Avron and Yoram Hirshfeld. On first order database query languages. In *LICS, July 15-18, 1991, Amsterdam, The Netherlands*, pages 226–231. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151647.

4 David A. Basin, Felix Klaedtke, Samuel Müller, and Eugen Zalinescu. Monitoring metric first-order temporal properties. *J. ACM*, 62(2):15:1–15:45, 2015. doi:10.1145/2699444.

5 Michael Benedikt and Leonid Libkin. Relational queries over interpreted structures. *J. ACM*, 47(4):644–680, 2000. doi:10.1145/347476.347477.

6 Achim Blumensath and Erich Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004. doi:10.1007/s00224-004-1133-y.

7 Sagar Chaki, Arie Gurfinkel, and Ofer Strichman. Decision diagrams for linear arithmetic. In *FMCAD, 15-18 November 2009, Austin, Texas, USA*, pages 53–60. IEEE, 2009. doi:10.1109/FMCAD.2009.5351143.

8 Jan Chomicki and David Toman. Implementing temporal integrity constraints using an active DBMS. *IEEE Trans. Knowl. Data Eng.*, 7(4):566–582, 1995. doi:10.1109/69.404030.

9 Jens Claußen, Alfons Kemper, Guido Moerkotte, and Klaus Peithner. Optimizing queries with universal quantification in object-oriented and object-relational databases. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB, August 25-29, 1997, Athens, Greece*, pages 286–295. Morgan Kaufmann, 1997. URL: http://www.vldb.org/conf/1997/P286.PDF.

10 E. F. Codd. Relational completeness of data base sublanguages. *Research Report / RJ / IBM / San Jose, California*, RJ987, 1972.

11 Erling Ellingsen. Regex golf, 2013. URL: https://alf.nu/RegexGolf.

12 Martha Escobar-Molano, Richard Hull, and Dean Jacobs. Safety and translation of calculus queries with scalar functions. In Catriel Beeri, editor, *PODS, May 25-28, 1993, Washington, DC, USA*, pages 253–264. ACM Press, 1993. doi:10.1145/153850.153909.

13 Allen Van Gelder and Rodney W. Topor. Safety and correct translation of relational calculus formulas. In Moshe Y. Vardi, editor, *PODS, March 23-25, 1987, San Diego, California, USA*, pages 313–327. ACM, 1987. doi:10.1145/28659.28693.

14 Allen Van Gelder and Rodney W. Topor. Safety and translation of relational calculus queries. *ACM Trans. Database Syst.*, 16(2):235–278, 1991. doi:10.1145/114325.103712.

15 Richard Hull and Jianwen Su. Domain independence and the relational calculus. *Acta Informatica*, 31(6):513–524, 1994. doi:10.1007/BF01213204.

16 Michael Kifer. On safety, domain independence, and capturability of database queries (preliminary report). In Catriel Beeri, Joachim W. Schmidt, and Umeshwar Dayal, editors, *Proceedings of the Third International Conference on Data and Knowledge Bases: Improving Usability and Responsiveness, June 28-30, 1988, Jerusalem, Israel*, pages 405–415. Morgan Kaufmann, 1988. doi:10.1016/b978-1-4832-1313-2.50037-8.

17 Nils Klarlund and Anders Møller. *MONA v1.4 User Manual*. BRICS, Department of Computer Science, University of Aarhus, January 2001. URL: http://www.brics.dk/mona/.

18 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

19 Hong-Cheu Liu, Jeffrey Xu Yu, and Weifa Liang. Safety, domain independence and translation of complex value database queries. *Inf. Sci.*, 178(12):2507–2533, 2008. doi:10.1016/j.ins.2008.02.005.

20 Jesper B. Møller. DDDLIB: A library for solving quantified difference inequalities. In Andrei Voronkov, editor, *CADE, July 27-30, 2002, Copenhagen, Denmark*, volume 2392 of *Lecture Notes in Computer Science*, pages 129–133. Springer, 2002. doi:10.1007/3-540-45620-1_9.

21 Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. Difference decision diagrams. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *CSL, September 20-25, 1999, Madrid, Spain*, volume 1683 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 1999. doi:10.1007/3-540-48168-0_9.

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

23 Jianmo Ni, Jiacheng Li, and Julian J. McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *EMNLP, November 3-7, 2019, Hong Kong, China*, pages 188–197. Association for Computational Linguistics, 2019. doi:10.18653/v1/D19-1018.

24 Robert A. Di Paola. The recursive unsolvability of the decision problem for the class of definite formulas. *J. ACM*, 16(2):324–327, 1969. doi:10.1145/321510.321524.

25 Martin Raszyk, David Basin, Srđan Krstić, and Dmitriy Traytel. Implementation, evaluation, and extended report associated with this paper, 2022. URL: https://github.com/rc2sql/rc2sql.

26 Peter Z. Revesz. *Introduction to Constraint Databases*. Texts in Computer Science. Springer, 2002. doi:10.1007/b97430.

27 Boris A Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *Doklady Akademii Nauk SSSR*, 70(4):569–572, 1950.

28 Moshe Y. Vardi. The decision problem for database dependencies. *Inf. Process. Lett.*, 12(5):251–254, 1981. doi:10.1016/0020-0190(81)90025-9.

29 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982. doi:10.1145/800070.802186.

30 Jun Yang. radb, 2019. URL: https://github.com/junyang/radb.

Characterising Fixed Parameter Tractability for Query Evaluation over Guarded TGDs

Cristina Feier ✉
University of Bremen, Germany

Abstract

We consider the parameterized complexity of evaluating Ontology Mediated Queries (OMQ) based on Guarded TGDs (GTGD) and Unions of Conjunctive Queries, in the case where relational symbols have unrestricted arity and where the parameter is the size of the OMQ. We establish exact criteria for fixed-parameter tractable (fpt) evaluation of recursively enumerable (r.e.) classes of such OMQs (under the widely held Exponential Time Hypothesis). One of the main technical tools introduced in the paper is an fpt-reduction from deciding parameterized uniform CSPs to parameterized OMQ evaluation. The reduction preserves measures known to be essential for classifying r.e. classes of parameterized uniform CSPs: submodular width (according to the well known result of Marx for unrestricted-arity schemas) and treewidth (according to the well known result of Grohe for bounded-arity schemas). As such, it can be employed to obtain hardness results for evaluation of r.e. classes of parameterized OMQs based on GTGD both in the unrestricted and in the bounded arity case. Previously, for bounded arity schemas, this has been tackled using a technique requiring full introspection into the construction employed by Grohe.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases omq, fpt evaluation, guarded tgds, unbounded arity, submodular width

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.12

Related Version *Full Version*: `https://arxiv.org/abs/2101.11727`

Acknowledgements I would like to thank the reviewers for the useful comments.

1 Introduction

Ontology mediated querying refers to the scenario where queries are posed to a database enhanced with a logical theory, commonly referred to as an *ontology*. The ontology refines the specific knowledge provided by the database by means of a logical theory. Popular ontology languages are decidable fragments of first order logic (FOL) like description logics [1], GTGD [11], or monadic disjunctive datalog [8]. Non-monotonic formalisms like answer set programming [28], or combinations of languages from the former and latter category like r-hybrid knowledge bases [31], g-hybrid knowledge bases [25], etc. have also been considered. As concerns query languages, atomic queries (AQs), conjunctive queries (CQs), and unions thereof (UCQs) are commonly used. An OMQ language is a tuple $(\mathcal{L}, \mathcal{Q})$, where $\mathcal{L}$ is an ontology language, and $\mathcal{Q}$ is a query language.

One thoroughly explored fragment of FOL as a basis for ontology specification languages is that of *tuple generating dependencies* (TGD). A tgd is a rule (logical implication) having as body and head conjunctions of atoms, where some variables occurring in head atoms might be existentially quantified (all other variables are universally quantified). As such, it potentially allows the derivation of atoms over fresh individuals (individuals not mentioned in the database). Answering even AQs with respect to sets of tgds is undecidable [11]. There have been lots of work on identifying decidable fragments [11, 2, 13]. A prominent such fragment is guarded TGDs (GTGD) [11]: a tgd is *guarded* if all universally quantified variables occur as terms of some body atom, called *guard*.

While query answering with respect to GTGD is decidable, the combined complexity of the problem is quite high: EXPTIME-complete for bounded arity schemas, and 2EXPTIME-complete in general. A natural question is when can OMQs from (GTGD, UCQ) be evaluated efficiently? A first observation is that by fixing the set of tgds, the complexity drops to NP for evaluating CQs, and to PTIME for evaluating AQs and CQs of bounded treewdith [12], which is similar to the complexity of query evaluation over databases [32].

Efficiency of query evaluation over databases has been a long evolving topic: starting with results concerning tractability of acyclic CQs evaluation [32], extended to bounded treewidth CQs in [15], and culminating, in the case of bounded arity schemas, with the result of Grohe which characterizes classes of CQs which can be efficiently evaluated in a parameterized complexity framework where the parameter is the query size. In this setting, under the assumption that FPT $\neq$ W[1], Grohe [24] establishes that exactly those r.e. classes of CQs which have bounded treewidth modulo homomorphic equivalence are fixed-parameter tractable. In this case fpt also coincides with polytime evaluation.

As concerns OMQs from (GTGD, UCQ) over bounded arity schemas, a similar characterization to that of Grohe has been established in a parameterized setting where the parameter is the size of the OMQ [3]. The cut-off criterium for efficient evaluation is again bounded treewidth modulo equivalence, only this time equivalence takes into account also the ontology. An OMQ from (GTGD, UCQ) has *semantic treewidth* k if there exists an equivalent OMQ from (GTGD, UCQ) whose UCQ has treewidth k [5]. Then, under the assumption that FPT $\neq$ W[1], a r.e. class of OMQs from (GTGD, UCQ) over bounded arity schemas can be evaluated in FPT iff it has bounded semantic treewidth. The similarity of the characterization with the database case is not coincidental: the results for OMQs build on the results of Grohe in a non-trivial way. In fact, the lower bound proof uses a central construction from [24], but has to employ sophisticated techniques to adapt this to OMQs.

The main open question which is adressed in this paper is: *when is it possible to efficiently evaluate OMQs from (GTGD, UCQ) in the general case, i.e. when there is no restriction concerning schema arity?* We again consider a parameterized setting, where the parameter is the size of the OMQ. This is a reasonable choice as the size of the OMQ is usually much smaller than the size of the database. As such, we are interested in *investigating the limits of fixed-parameter tractability of evaluating a class of OMQs* $\mathbb{Q}$ *from (GTGD, UCQ)*. We denote such a parameterized problem as p-OMQ($\mathbb{Q}$).

Before stating our main results, we review some results concerning efficiency of solving *constraint satisfaction problems* (CSP). These are relevant as CQ evaluation over databases is tightly linked to solving so-called *uniform CSPs*. Given classes of relational structures $\mathbb{A}$ and $\mathbb{B}$, a CSP problem $(\mathbb{A}, \mathbb{B})$ asks whether there exists a homomorphism from some relational structure in $\mathbb{A}$ to another relational structure in $\mathbb{B}$. In the uniform case, $\mathbb{A}$ is fixed, $\mathbb{B}$ is the class of all relational structures and the problem is denoted as $(\mathbb{A}, \_)$. The parameterized version of the problem (where for a problem instance (A, B) the parameter is the size of A) is denoted as p-CSP$(\mathbb{A}, _)$. When restricted to classes of finite structures, uniform CSPs can be seen as an alternative presentation of the problem of evaluating a class of Boolean CQs over databases. In fact, Grohe's characterization for fpt evaluation of r.e. classes of CQs in the bounded arity case has been achieved via a uniform CSP detour [24].

In the unrestricted arity case, Marx [30] established in a seminal result the border for fpt evaluation of uniform CSPs of the form p-CSP$(\mathbb{A}, _)$, where $\mathbb{A}$ is closed under underlying hypergraphs. The restriction has been lifted in [16], yielding a full characterization for parameterized uniform CSPs of unrestricted arity. Both results are based on a widely held conjecture, the Exponential Time Hypothesis [26] and rely on a new structural measure, *submodular width*. The measure will also play a major role in our characterizations:

Main Result 1. Let $\mathbb{Q}$ be a r.e. class of OMQs from (GTGD, UCQ). Assuming the Exponential Time Hypothesis, p-OMQ($\mathbb{Q}$) is fixed-parameter tractable iff $\mathbb{Q}$ has bounded semantic submodular width.

To prove the result, we exploit the fact that every OMQ from (GTGD, UCQ) can be rewritten into an OMQ from (GDLog, UCQ) [3], where GDLog stands for Guarded Datalog, the restriction of GTGD to rules with only universally quantified variables. For OMQs from (GDLog, UCQ), we construct equivalent OMQs called *covers* which are witnesses for bounded semantic submodular width. Covers are based on sets of *characteristic databases for OMQs* which are databases that entail the OMQs and which are small with respect to the homomorphism order, in a very specific sense. While in a database setting, the database induced by a CQ can be seen as a canonical database which entails the query, in the case of OMQs which pose restrictions on the database schema this might not be possible: a CQ in an OMQ might contain symbols which are not allowed to occur in a database. In fact, this is a typical usage of ontologies: to enrich the database schema with new terminology. As such, we identify other representative databases for OMQs. Based on these notions, for OMQs from (GDLog, UCQ), we establish the following:

Main Result 2. For $\mathbb{Q}$ a r.e. class of OMQs from (GDLog, UCQ), let $\mathbb{Q}_c$ and $\mathbb{D}_{\mathbb{Q}}$ be the classes of covers and characteristic databases for OMQs from $\mathbb{Q}$, respectively. Under the Exponential Time Hypothesis, the following statements are equivalent:

1. p-OMQ($\mathbb{Q}$) is fixed-parameter tractable;
2. $\mathbb{Q}_c$ has bounded submodular width;
3. $\mathbb{D}_{\mathbb{Q}}$ has bounded submodular width.

The hardness result for the above characterization is obtained via an fpt-reduction from parameterized uniform CSP evaluation to parameterized OMQ evaluation.

Main Result 3. For $\mathbb{Q}$ a r.e. class of OMQs from (GDLog, UCQ), there exists an fpt-reduction from p-CSP($\mathbb{D}_{\mathbb{Q}}$, _) to p-OMQ($\mathbb{Q}$).

The reduction is important as a stand-alone result as it enables lifting of other hardness complexity results from the database world to the OMQ one. It enables us to obtain analogues of our Main Result 1 and Main Result 2 for the case of OMQs based on GTGD with bounded arity schemas using the results from [24] as a black-box. In this case, submodular width is replaced with treewidth and the Exponential Time Hypothesis with the assumption that FPT $\neq$ W[1]. As such, we re-establish the semantic characterization from [3], and we provide an alternative characterization for fpt evaluation of OMQs from (GDLog, UCQ) over bounded arity schemas.

2 Preliminaries

Structures, Databases. A *schema* $\mathbf{S}$ is a finite set of relation symbols with associated arities. An $\mathbf{S}$-*fact* has the form $r(\mathbf{a})$, where $r \in \mathbf{S}$, and $\mathbf{a}$ is a tuple of constants of size the arity of r. An $\mathbf{S}$-*structure* A is a set of $\mathbf{S}$-facts. The domain of a structure A, $\mathsf{dom}(A)$, is the set of constants which occur in facts in A. Given a structure A and a subset $C \subseteq \mathsf{dom}(A)$, the *sub-structure of A induced by C*, $A|_C$,it is the structure containing all facts $r(\mathbf{b}) \in A$ such that $\mathbf{b} \subseteq C$. *The product of two structures A and B*, $A \times B$, is a structure with domain $\mathsf{dom}(A) \times \mathsf{dom}(B)$ consisting of all facts of the form $r((a_1, b_1), \ldots, (a_n, b_n))$, where $r(a_1, \ldots, a_n) \in A$ and $r(b_1, \ldots, b_n) \in B$. Given two structures A and B, a function $f : \mathsf{dom}(A) \to \mathsf{dom}(B)$ is said to be a *homomorphism* from A to B, if for every fact $r(\mathbf{a}) \in A$,

there exists a fact $r(\mathbf{b}) \in B$ such that $f(\mathbf{a}) = \mathbf{b}$. The image of A in B under f, $f(A)$, is the set of facts of the form $r(f(\mathbf{a}))$ in B, where $r(\mathbf{a})$ is from A. When such a homomorphism exists we say that A *maps into* B, denoted $A \to B$.

Two structures are *equivalent* if $A \to B$ and $B \to A$. We write $A \leftrightarrow B$. They are *isomorphic* if there exists a homomorphism h from A to B which is bijective and onto, i.e. for every fact $r(\mathbf{b}) \in B$, there exists a fact $r(\mathbf{a}) \in A$ such that $h(\mathbf{a}) = \mathbf{b}$. A structure A is a *core* if every homomorphism from A to itself is injective. Every structure A has a sub-structure A' which is equivalent to A and is a core. All cores of a structure are isomorphic. The homomorphism relation $\to$ is a pre-order over the set of all structures. When restricted to structures which are cores (taken up to isomorphism), $\to$ is a partial order. Following [21], we will refer to this order as the *homomorphism order*.

An $\mathbf{S}$-*database* is a finite $\mathbf{S}$-structure. For D a database, and $\mathbf{a} \subseteq \mathsf{dom}(D)$, $\mathbf{a}$ is a *guarded set in* D if there exists a fact $r(\mathbf{a}')$ in D such that $\mathbf{a} \subseteq \mathbf{a}'$. It is a *maximal guarded set* if there exists no guarded strict superset. As in the previous definitions, we will sometimes abuse notation by using tuples of constants to refer to the underlying sets of constants instead.

Conjunctive Queries, Atomic Queries. A *conjunctive query (CQ)* is a formula of the form $q(\mathbf{x}) = \exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$, with $\mathbf{x}$ and $\mathbf{y}$ tuples of variables and $\phi(\mathbf{x}, \mathbf{y})$ a conjunction of atoms having as terms only variables from $\mathbf{x} \cup \mathbf{y}$. The set $\mathbf{x}$ is the set of *answer variables* of q, while the set $\mathbf{y}$ is the set of *existential variables* of q. We denote with $\mathsf{var}(q)$ the set of variables of q, and with $D[q]$ the *canonical database of* q, i.e. the set of atoms which occur in ϕ (viewed as facts). When $\mathbf{x}$ is empty, the CQ is said to be *Boolean* (BCQ). We will sometimes use BCQs or their canonical databases interchangeably. A sub-query of a BCQ q is a BCQ p such that $D[p] \subseteq D[q]$. A *union of conjunctive queries (UCQ)* is a formula of the form $q(\mathbf{x}) = q_1(\mathbf{x}) \vee \ldots \vee q_n(\mathbf{x})$, where each $q_i(\mathbf{x})$ is a CQ, for $i \in [n]$. An *atomic query (AQ)* is a CQ in which ϕ contains a single atom. In the following, whenever we refer to CQs or UCQs, we tacitly assume they are Boolean. As concerns AQs, unless stated otherwise, we assume they are of the form $r(\mathbf{x})$, i.e. they contain no existentially quantified variables.

For a structure I, a CQ $q(\mathbf{x})$, and a tuple of constants $\mathbf{a}$ from $\mathsf{dom}(I)$, $\mathbf{a}$ *is an answer to* q *over* I, or $I \models q(\mathbf{a})$, if there is a homomorphism h from $D[q]$ to I such that $h(\mathbf{x}) = \mathbf{a}$. If $q(\mathbf{x})$ is a UCQ of the form $q_1(\mathbf{x}) \vee \ldots \vee q_n(\mathbf{x})$, $I \models q(\mathbf{a})$ if $I \models q_i(\mathbf{a})$, for some $i \in [n]$.

Ontology Mediated Queries. An *ontology mediated query (OMQ)* Q is a triple $(\mathcal{O}, \mathbf{S}, q(\mathbf{x}))$, where $\mathcal{O}$ is an ontology, $\mathbf{S}$ is a schema, and $q(\mathbf{x})$ is a query. When $\mathcal{O}$ is specified using the ontology language $\mathcal{L}$, and q using the query language $\mathcal{Q}$, we say that Q belongs to the OMQ language $(\mathcal{L}, \mathcal{Q})$. The schema $\mathbf{S}$ specifies which relational symbols can occur in databases over which Q is evaluated. We say that Q is an $\mathbf{S}$-*OMQ*.

Given an $\mathbf{S}$-database D, and a tuple of constants $\mathbf{a}$, all of which are from $\mathsf{dom}(D)$, we say that $\mathbf{a}$ is an *answer to* Q *over* D, or $D \models Q(\mathbf{a})$, if $\mathcal{O} \cup D \models q(\mathbf{a})$, where $\models$ is the entailment relation in $\mathcal{L}$. For Q_1 and Q_2 two OMQs over the same schema $\mathbf{S}$, we say that Q_1 is *contained in* by Q_2, written $Q_1 \subseteq Q_2$, if for every $\mathbf{S}$-database D and tuple of constants $\mathbf{a}$: $D \models Q_1(\mathbf{a})$ implies $D \models Q_2(\mathbf{a})$. We also say that Q_1 is *equivalent* to Q_2 if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

TGDs, Guarded TGDs. A *tuple generating dependency* (tgd) is a first order sentence of the form $\forall \mathbf{x} \forall \mathbf{y}\ \phi(\mathbf{x}, \mathbf{y}) \to \exists \mathbf{z}\ \psi(\mathbf{x}, \mathbf{z})$ (abbreviated $\phi(\mathbf{x}, \mathbf{y}) \to \exists \mathbf{z}\ \psi(\mathbf{x}, \mathbf{z})$), with ϕ, the *body* of the tgd, and ψ, the *head* of the tgd, being conjunctions of atoms with terms from $\mathbf{x} \cup \mathbf{y}$, and from $\mathbf{x} \cup \mathbf{z}$, resp. A set of tgds is a *TGD program*. The problem of *evaluating a UCQ* $q(\mathbf{x})$ *over a TGD program* $\mathcal{O}$ *w.r.t. an* $\mathbf{S}$-*database* D consists in checking whether for some tuple

$\mathbf{a}$ over $\mathsf{dom}(D)$, it is the case that $D \models Q(\mathbf{a})$, where Q is the OMQ $(\mathcal{O}, \mathbf{S}, q(\mathbf{x}))$. While the problem is undecidable, [11], it can be characterized via a completion of the database D to a structure which is a universal model of $\mathcal{O}$ and D, called *chase* [29, 19, 27]. We will denote with $\mathsf{ch}_{\mathcal{O}}(D)$ the chase of $\mathcal{O}$ w.r.t. D. Then, $D \models Q(\mathbf{a})$ iff $\mathsf{ch}_{\mathcal{O}}(D) \models q(\mathbf{a})$.

There are several variants of the chase; here we describe the *oblivious chase*. Let $(\mathsf{ch}_k(\mathcal{O}, D))_{k\geq 0}$ be a sequence of structures such that $\mathsf{ch}_0(\mathcal{O}, D) = D$. Then, for every $i > 0$, $\mathsf{ch}_i(\mathcal{O}, D)$ is obtained from $\mathsf{ch}_{i-1}(\mathcal{O}, D)$ by considering all homomorphisms h from the body of some tgd $\phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\psi(\mathbf{x}, \mathbf{z})$ in $\mathcal{O}$ to $\mathsf{ch}_{i-1}(\mathcal{O}, D)$ s.t. at least one atom from ϕ is mapped by h into a fact from $\mathsf{ch}_{i-1}(\mathcal{O}, D) \setminus \mathsf{ch}_{i-2}(\mathcal{O}, D)$, and adding to $\mathsf{ch}_i(\mathcal{O}, D)$ all facts obtained from atoms in $\psi(\mathbf{x}, \mathbf{z})$ by replacing each $x \in \mathbf{x}$ with $h(x)$ and each $z \in \mathbf{z}$ with some fresh constant. Then, $\mathsf{ch}_{\mathcal{O}}(D) = \bigcup_{k\geq 0} \mathsf{ch}_k(\mathcal{O}, D)$. Note that $\mathsf{ch}_{\mathcal{O}}(D)$ might be infinite.

A tgd for which all universally quantified variables occur in a body atom is *guarded*. A *GTGD program* is a set of guarded tgds. Unlike evaluation of OMQs based on unrestricted tgds, evaluation of OMQs from (GTGD, UCQ) is decidable [10]. A GTGD program in which all tgds have universally quantified variables only is a *guarded Datalog (GDLog)* program. For every GDLog ontology $\mathcal{O}$ and every database D, $\mathsf{ch}_{\mathcal{O}}(D)$ is finite; furthermore, for every fact $r(\mathbf{a}) \in \mathsf{ch}_{\mathcal{O}}(D)$, there exists a guarded set $\mathbf{a}'$ over D such that $\mathbf{a} \subseteq \mathbf{a}'$.

Parameterized Complexity. For Σ some finite alphabet, a *parameterized problem* is a tuple (P, κ), where $P \subseteq \Sigma^*$ is a problem, and $\kappa : \Sigma^* \rightarrow \mathbb{N}$ is a PTIME computable function called the *parameterization* of P. Such a parameterized problem is *fixed-parameter tractable* if there exists an algorithm for deciding P for an input $x \in \Sigma^*$ in time $f(\kappa(x))poly(|x|)$, where f is a computable function and *poly* is a polynomial. The class of all fixed-parameter tractable problems is denoted as FPT.

Given two parameterized problems (P_1, κ_1) and (P_2, κ_2) over alphabets Σ_1 and Σ_2, an *fpt-reduction* from (P_1, κ_1) to (P_2, κ_2) is a function $R : \Sigma_1^* \rightarrow \Sigma_2^*$ with the following properties:

1. $x \in P_1$ iff $R(x) \in P_2$, for every $x \in \Sigma_1^*$,
2. there exists a computable function f such that $R(x)$ is computable in time $f(\kappa_1(x))poly(|x|)$,
3. there exists a computable function g such that $\kappa_2(R(x)) \leq g(\kappa_1(x))$, for all $x \in \Sigma_1^*$.

Downey and Fellows [20] defined a hierarchy of parameterized complexity classes $\mathrm{W}[0] \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \ldots$, where $\mathrm{W}[0] = \mathrm{FPT}$ and each inclusion is believed to be strict. Each class W[i], with $i \geq 0$, is closed under fpt-reductions.

A class of interest for us is W[1] as under the assumption that $\mathrm{FPT} \neq \mathrm{W}[1]$, it is possible to establish intractability results (non-membership to FPT) for parameterized problems. A well-known W[1]-complete problem is the parameterized k-clique problem, where the parameter is k: for an input (G, k), with G a graph and $k \in \mathbb{N}^*$, it asks whether G has a k-clique. A stronger assumption than $\mathrm{FPT} \neq \mathrm{W}[1]$, which is widely believed to hold and can be used to establish intractability results, is the *Exponential Time Hypothesis*: it states that 3-SAT with n variables cannot be decided in $2^{o(n)}$ time [26].

Structural Measures: Treewdith, Submodular Width. A *hypergraph* is a pair $H = (V, E)$ with V a set of *nodes* and $E \subseteq 2^V \setminus \{\emptyset\}$ a set of *edges*. A *tree decomposition* of H is a pair $\delta = (T_\delta, \chi)$, with $T_\delta = (V_\delta, E_\delta)$ a tree, and χ a labeling function $V_\delta \rightarrow 2^V$ such that:

1. $\bigcup_{t\in V_\delta} \chi(t) = V$.
2. If $e \in E$, then $e \subseteq \chi(t)$ 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 *treewidth* of H, $\mathsf{TW}(H)$, is the smallest k such that there exists a tree decomposition (T_δ, χ) of H, with $T_\delta = (V_\delta, E_\delta)$, such that for every $t \in V_\delta$, $|\chi(t)| \leq k$. A function $f : 2^V \to \mathbb{R}_{\geq 0}$ is *submodular* if $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$. It is *edge-dominated* if $f(e) \leq 1$ for all $e \in E$. The *submodular width* of H, $\mathsf{SMW}(H)$, is the smallest k such that for every monotone submodular edge-dominated function f, for which $f(\emptyset) = 0$, there exists a tree decomposition (T_δ, χ) of H, with $T_\delta = (V_\delta, E_\delta)$, such that $f(\chi(t)) \leq k$ for all $t \in V_\delta$.

Every structure I has an associated hypergraph $H_I = (V, E)$, with $V = \mathsf{dom}(I)$ and E the set of guarded sets in I. A tree decomposition $\delta = (T_\delta, \chi)$ of I, with $T = (V_\delta, E_\delta)$, is a tree decomposition of H_I. We say that δ is *guarded* if for every $v \in V_\delta$, $\chi(v)$ is a guarded set in I: $\chi(v) \in E$. Let X range over $\{\mathsf{TW}, \mathsf{SMW}\}$. For a structure I, $\mathsf{X}(I) = \mathsf{X}(H_I)$, for a CQ q, $\mathsf{X}(q) = \mathsf{X}(H_q)$, while for a UCQ q', $\mathsf{X}(q') = \mathsf{max}_q$ is a CQ in $_{q'}(X(q))$. Finally, for an OMQ $Q = (\mathcal{O}, \mathbf{S}, q)$, with q a UCQ, $\mathsf{X}(Q) = \mathsf{X}(q)$.

3 Normalizing OMQs: Characteristic Databases and Covers

As seen earlier, *equivalence-based measures* frequently play a role in fpt characterizations. Following [6, 4, 3], we refer to such measures as *semantic measures*. In particular, for an OMQ $Q \in (\mathcal{L}, \mathrm{UCQ})$, with $\mathcal{L}$ an ontology language, and a structural measure $\mathsf{X} \in \{\mathsf{TW}, \mathsf{SMW}\}$, the semantic X-width of Q is the smallest k such that there exists an OMQ Q' from $(\mathcal{L}, \mathrm{UCQ})$ with $Q \equiv Q'$ and $\mathsf{X}(Q') = k$. A class $\mathbb{Q}$ of OMQs has *bounded semantic* X*-width* if there exists $k > 0$ such that every OMQ in $\mathbb{Q}$ has semantic X-width at most k. However, in order to establish such characterizations, an important issue is finding witnesses of (bounded) semantic measures, i.e. classes of OMQs of low measures which are equivalent to the original ones.

For CQs, cores serve as witnesses for semantic treewidth [24] and also for semantic submodular width [16]. Thus, as concerns classes of CQs of bounded semantic X-width, with $\mathsf{X} \in \{\mathsf{TW}, \mathsf{SMW}\}$, the class of cores of CQs from the original class serves as a witness, i.e. it has actual bounded X-width. As concerns OMQs based on UCQs, resorting to cores of CQs in UCQs does not necessarily lead to witnesses of low width. The ontology also plays a role in lowering semantic measures. For examples of this phenomenon as concerns semantic treewidth, see [4, 3]. Here, we show how the ontology influences semantic submodular width:

▶ **Example 1.** For R a binary relational symbol, $i \in \mathbb{N}$, with $i > 1$, and $\mathbf{x}_i$ an i-tuple of variables, we denote with $\psi_i^R(\mathbf{x}_i)$ the formula $R(x_1, x_2) \wedge R(x_1, x_3) \wedge \cdots \wedge R(x_{i-1}, x_i)$, i.e. the hypergraph associated to ψ_i^R is the i-clique. Let T be a binary relational symbol and $\mathbb{Q}$ be the class of OMQs $(Q_i)_{i>1}$, with $Q_i = (\mathcal{O}_i, \mathbf{S}_i, q_i)$, where:

$$\mathcal{O}_i = \{S_i(\mathbf{x}_i) \to \psi_i^R(\mathbf{x}_i)\} \qquad \mathbf{S}_i = \{S_i, T\} \qquad q_i = \exists \mathbf{x}_i \; \psi_i^R(\mathbf{x}_i) \wedge \psi_i^T(\mathbf{x}_i)$$

Then $\mathbb{Q}$ has unbounded submodular width. As for every $i > 1$, $H_{D[q_i]}$ is the i-clique, every tree decomposition (T, χ) of $H_{D[q_i]}$, with $T = (V, E)$, must contain some node $t \in V$ such that $\chi(t) = \mathbf{x}_i$. Let $f : 2^{\mathbf{x}_i} \to \mathbb{R}_{\geq 0}$ be the monotone submodular function $f(X) = |X|/2$. Then f is also edge-dominated with respect to $D[q_i]$, and its minimum over all tree decompositions of $H_{D[q_i]}$ is $i/2$. Thus, $\mathsf{SMW}(Q_i) \geq i/2$, for every $i > 1$.

On the other hand, for every $i > 1$, R is not part of the schema $\mathbf{S}_i$ and the only way to derive it is using the unique tgd from $\mathcal{O}_i$. Thus, every $\mathbf{S}_i$-database D_i such that $D_i \models Q_i$ must contain an atom of the form $S_i(\mathbf{c}_i)$, where $\mathbf{c}_i$ is an i-tuple of constants. Then, for every $i > 0$, Q_i is equivalent to the OMQ $Q_i' = (\mathcal{O}_i, \mathbf{S}_i, q_i')$, with $q_i' = \exists \mathbf{x}_i \; S_i(\mathbf{x}_i) \wedge \psi_i^R(\mathbf{x}_i) \wedge \psi_i^T(\mathbf{x}_i)$.

Let $\mathbb{Q}'$ be the class of OMQs $(Q'_i)_{i>1}$. As for every $i > 1$, q'_i is guarded, i.e. it contains an atom $S_i(\mathbf{x}_i)$ which has as terms $\mathsf{var}(q'_i)$, it follows that $\mathsf{SMW}(q'_i) \leq 1$: this is due to the fact that only edge-dominated functions are considered when defining the submodular width and thus the guard ensures that for every such function f, $f(\mathbf{x}_i) \leq 1$. Thus, $\mathbb{Q}'$ has bounded submodular width and $\mathbb{Q}$ has bounded semantic submodular width.

Example 1 shows that submodular width can be lowered by adding extra atoms to CQs in the original OMQ. In this section, we normalize OMQs from (GDLog, UCQ) by extending (images of) CQs in the original OMQ with facts occurring in databases which entail the OMQ. We obtain equivalent OMQs called *covers*. The purpose of the added atoms is to provide guards for atoms in CQs and to potentially lower submodular width, as in Example 1. Intuitively, by adding such guards, the submodular width is decreased as the space of edge-dominated submodular functions is shrunk.

At the same time, when constructing covers, we do not want to add too many atoms: adding cliques of unbounded size (without a guard) would obviously not decrease submodular width. As such, we use as the basis for the construction only a set of selected databases which are small w.r.t. the homomorphism order and which we call *extended characteristic databases*. Ideally, we would consider only databases which entail the OMQ and which are minimal w.r.t. the homomorphism order. However, for a given OMQ there might be no such minimal databases:

▶ **Example 2.** Let $Q = (\mathcal{O}, \mathbf{S}, q)$ be the following OMQ from (GDLog, UCQ):

$$\mathcal{O} = \{A(x) \wedge R(x,y) \to A(y),\ A(x) \to B(x)\}$$
$$\mathbf{S} = \{A, R, C\}$$
$$q = \exists x\ B(x) \wedge R(x,x) \wedge C(x)$$

Also, for every $n \in \mathbf{N}$, let D_n be the $\mathbf{S}$-database $\{A(x_0), C(x_n), R(x_n, x_n)\} \cup \{R(x_i, x_{i+1}) \mid 0 \leq i < n\}$. Then, for every $n \in \mathbf{N}$, $D_n \models Q$, D_n is a core, $D_{n+1} \to D_n$, and $D_n \not\to D_{n+1}$. Thus, D_n is not minimal w.r.t. $\to$. Furthermore, for every $\mathbf{S}$-database D, it can be checked that $D \models Q$ implies that there exists $n \in \mathbf{N}$ such that $D_n \to D$, and thus D is not minimal w.r.t. the homomorphism order.

It is also not possible to use the set of canonical databases of CQs occurring in the OMQ as a basis for the construction, as in the presence of schema restrictions, those databases might not be over the expected schema. As such, to identify a set of selected databases, we apply successive transformations on the set of databases which entail an OMQ. The transformations are described in Sections 3.1, 3.2, and 3.3. Section 3.4 brings all the concepts together to define (extended) characteristic databases and covers.

3.1 Query Initial Databases

In our quest to refine databases which entail an OMQ we look at how CQs from the OMQ map into the chase of a given database. In the following, a *contraction* of a CQ q is a CQ obtained from q by variable identification.

▶ **Definition 3.** *For $Q = (\mathcal{O}, \mathbf{S}, q)$ from (GDLog, UCQ) and D an $\mathbf{S}$-database such that $D \models Q$, we say that D is* query-initial (qi) w.r.t. *Q if for every $\mathbf{S}$-database D' such that $D' \to D$ and $D' \models Q$, and every contraction p of some CQ in q it is the case that: $p \to \mathsf{ch}_{\mathcal{O}}(D')$ iff $p \to \mathsf{ch}_{\mathcal{O}}(D)$.*

Thus, qi databases entail an OMQ and are minimal w.r.t. the set of contractions which map into their chase. They are related to injectively only databases which have been introduced in [4]. By a simple induction argument, it can be shown that:

▶ **Lemma 4.** *Let Q be an $\mathbf{S}$-OMQ from (GDLog, UCQ), and D an $\mathbf{S}$-database with $D \models Q$. If D is qi w.r.t. Q, then for every database $D' \to D$, $D' \models Q$ implies D' is qi w.r.t. Q. If D is not qi w.r.t. Q, there exists a database $D' \to D$ such that $D' \models Q$ and D' is qi w.r.t. Q.*

▶ **Example 5.** Let $Q = (\mathcal{O}, \mathbf{S}, q)$ be the OMQ with:

$$\mathcal{O} = \{U(x,y,z) \wedge V(x,z) \to T(x,z), W(x,y,z) \to S(y,z)\}$$
$$\mathbf{S} = \{R, U, V, W\}$$
$$q = \exists x, y, z \; R(x,y) \wedge S(y,z) \wedge T(z,x)$$

Also, let D_1 and D_2 be the $\mathbf{S}$-databases: $D_1 = \{R(a,b), W(d,b,a), U(a,d,a), V(a,a)\}$ and $D_2 = \{R(a,b), W(d,b,c), U(c,d,a), V(c,a)\}$. Then $D_2 \to D_1$.

Let q' be the following contraction of q: $q' = \exists x, y \; R(x,y) \wedge S(y,x) \wedge T(x,x)$ (obtained by identification of x and z in q). Then both q and q' map into $\mathsf{ch}_{\mathcal{O}}(D_1)$, but only q maps into $\mathsf{ch}_{\mathcal{O}}(D_2)$. Thus, D_1 is not minimal w.r.t the set of contractions which map into its chase, and consequently it is not qi w.r.t. Q. However, D_2 is qi w.r.t. Q.

3.2 Guarded Unravelings

Another concept which will turn useful for defining characteristic databases is that of *guarded unraveling*. The operation was first introduced in [22]. Given a database D and a guarded set set $\mathbf{a}$ over D, it unfolds D into a (potentially infinite) structure $I^{\mathbf{a}}$, the *guarded unraveling of D at $\mathbf{a}$*, which admits a guarded tree decomposition $\delta = (T, \chi)$, with $T = (V, E)$. The vertices $t \in V$ of T are sequences of the form $\mathbf{a}_0 \dots \mathbf{a}_n$, where $\mathbf{a}_0 = \mathbf{a}$, and $\mathbf{a}_1, \dots, \mathbf{a}_n$ are maximal guarded sets in D such that $\mathbf{a}_i \cap \mathbf{a}_{i+1} \neq \emptyset$, and $\mathbf{a}_i \neq \mathbf{a}_{i+1}$, for every $0 \leq i < n$. For every $t_1, t_2 \in V$, $(t_1, t_2) \in E$ iff $t_2 = t_1\mathbf{b}$, for some maximal guarded set $\mathbf{b}$ in D.

Intuitively, we unravel D guided by δ: each vertex $t \in V$ of the form $\mathbf{a}_0 \dots \mathbf{a}_n$ can be seen as a local representative (in the unraveling) of the the maximal guarded set $\mathbf{a}_n$ in D; as such, $\chi(t)$ contains copies of the constants from $\mathbf{a}_n$. Assuming we build T in an inductive fashion, we set $\chi(r) = \mathbf{a}$, where r is the root of T and then, for each newly created $t \in V$ of the form $\mathbf{a}_1, \dots, \mathbf{a}_n$, we introduce fresh copies of constants a which occur in the guarded set $\mathbf{a}_n$ corresponding to t, but do not occur in the guarded set $\mathbf{a}_{n-1}$ corresponding to its predecessor node $t' = \mathbf{a}_1 \dots \mathbf{a}_{n-1}$. We denote such a fresh copy of a, introduced while creating t, as a_t. Then, we set $\chi(t) = \{a_{t'} \mid a \in \mathbf{a}_n \cap \mathbf{a}_{n-1}\} \cup \{a_t \mid a \in \mathbf{a}_n \setminus \mathbf{a}_{n-1}\}$. We also define D_t to be a copy of the database $I|_{\mathbf{a}_n}$, where every $a \in \mathbf{a}_n$ has been replaced with the corresponding a_t or $a_{t'}$ from $\chi(t)$. $I^{\mathbf{a}}$ is the union of all databases D_t with $t \in V$.

Guarded unravelings have the property that for every OMQ Q' from (GDLog, AQ), every database D, every guarded set $\mathbf{a}$ in D, and every tuple $\mathbf{a}' \subseteq \mathbf{a}$: $D \models Q'(\mathbf{a}')$ implies $I^{\mathbf{a}} \models Q'(\mathbf{a}')$ [22]. It can also easily be seen that, due to the existence of a guarded tree decomposition, the submodular width of $I^{\mathbf{a}}$ is 1. As such, for a given OMQ $Q = (\mathcal{O}, \mathbf{S}, q)$ from (GDLog, UCQ) and some $\mathbf{S}$-database D such that $D \models Q$, we will use them to disentangle parts of D which are needed to entail specific atoms from some CQ in Q, i.e. we will replace some parts of D with corresponding guarded unravelings.

In general, for a guarded set $\mathbf{a}$ in D, $I^{\mathbf{a}}$ is infinite and thus cannot be used straightaway. By compactness, there exists a finite subset (a database) $D^{\mathbf{a}}$ which fulfills the same property w.r.t. entailment of OMQs based on AQs. We fix such a database $D^{\mathbf{a}}$. However, as we

want to unravel databases which entail OMQs as much as possible, we place a stronger requirement on the chosen $D^{\mathbf{a}}$: for every Boolean sub-query p' of some CQ p in q, whenever $I^{\mathbf{a}} \models (\mathcal{O}, \mathbf{S}, p')$, it must be the case that $D^{\mathbf{a}} \models (\mathcal{O}, \mathbf{S}, p')$. As this requirement is relative to Q, we will refer to $D^{\mathbf{a}}$ as the *guarded unraveling of D at $\mathbf{a}$ w.r.t. Q*. However, Q will be clear in most cases from the context so it will be omitted.

3.3 Diversifications

A *diversification* of a database is essentially a database which maps into the original one in a specific way. For a function f and S a subset of its domain, we denote with $f|_S$ the restriction of f on S. Also, we denote with $\mathsf{ran}(f)$ the range of f. A homomorphism h from a structure A to a structure B is said to be *injective on guarded sets (i.g.s.)* if $h|_{\mathbf{a}}$ is injective, for every guarded set $\mathbf{a}$ in A. For a database D, a constant $c \in \mathsf{dom}(D)$ is *isolated in D* if it occurs in a single fact in D. The *kernel* of a database D, $\mathsf{ker}(D)$, is the set of non-isolated constants in D.

▶ **Definition 6.** *A* diversification *of a database D_0 is a tuple $(D, \uparrow)$, where D is a database which maps into D_0 via the homomorphism $\uparrow$ such that:*
1. *$\uparrow_{|\mathsf{ker}(D)}$ is injective, and*
2. *$\uparrow$ is i.g.s.*

We write $D \preceq D_0$, whenever there exists a diversification $(D, \uparrow)$ of D_0.

▶ **Example 7.** Let $D_2 = \{R(a,b), W(d,b,c), U(c,d,a), V(c,a)\}$ be the database from Example 5. Also, let D be the $\mathbf{S}$-database: $\{R(a,b), W(d_1,b,c), U(c,d_2,a)\}$. Then, the mapping $\uparrow\colon \mathsf{dom}(D) \to \mathsf{dom}(D_2)$, which is the identity on $\{a,b,c\}$ and maps d_1 and d_2 to d is a homomorphism from D to D_2. As $\mathsf{ker}(D) = \{a,b,c\}$ and $\uparrow$ is i.g.s., it follows that $(D, \uparrow)$ is a diversification of D_2 and thus $D \preceq D_2$.

For a database D which entails an OMQ Q, we will use diversifications of D in conjunction with guarded unravelings of D w.r.t. Q to construct new databases which map into D and which still entail Q. We start with a construction which glues guarded unravelings of some database D_0 w.r.t. Q to a database D which maps into D_0 via some homomorphism $\uparrow$ which is i.g.s. (in this case $(D, \uparrow)$ need not be a diversification of D_0). We denote with $\mathsf{ext}_Q(D, \uparrow, D_0)$ the database obtained from D by adding for each maximal guarded set $\mathbf{a}$ in D the database $D_0^{\mathbf{a}}$ obtained from the guarded unraveling $D_0^{\uparrow(\mathbf{a})}$ of D_0 w.r.t. Q by renaming the constants in $\uparrow(\mathbf{a})$ to those in $\mathbf{a}$. When Q is clear from the context, we write $\mathsf{ext}(D, \uparrow, D_0)$.

▶ **Definition 8.** *For Q an $\mathbf{S}$-OMQ from (GDLog, UCQ) and D_0 an $\mathbf{S}$-database with $D_0 \models Q$:*
1. *$\mathsf{div}(D_0, Q)$ is the set of diversifications $(D, \uparrow)$ of D_0 for which $\mathsf{ext}(D, \uparrow, D_0) \models Q$;*
2. *a diversification $(D, \uparrow)$ from $\mathsf{div}(D_0, Q)$ is* minimal w.r.t. Q *if D is a core and there is no other diversification $(D', \downarrow)$ from $\mathsf{div}(D_0, Q)$ such that $D' \preceq D$, and $D \not\preceq D'$;*
3. *$\mathsf{mdiv}(D_0, Q)$ is the set of all minimal diversifications of D_0 w.r.t. Q.*

Intuitively, for a minimal diversification $(D, \uparrow)$ of D_0 w.r.t. Q and the ensuing database $\mathsf{ext}(D, \uparrow, D_0)$, the D-part of $\mathsf{ext}(D, \uparrow, D_0)$ is important for preserving some part of the hypergraph of D_0 needed to satisfy the geometry (hypergraph) of a query. Guarded unravelings provide the necessary information to entail specific atoms in the query.

▶ **Example 9.** Let $Q = (\mathcal{O}, \mathbf{S}, q)$ and D_2 be as in Example 5. Also let $(D, \uparrow)$ be the diversification of D_2 introduced in Example 7 and let $D^+ = \mathsf{ext}(D, \uparrow, D_2)$ be the database obtained from D by adding guarded unravelings of D_2. We do not explicitly construct the guarded unravelings, but note that $D_2^{(a,c,d)}$ will contain the fact $V(c,a)$. Let $D' = D \cup \{V(c,a)\}$. Then $D' \subseteq D^+$ and $D' \models Q$, thus $D^+ \models Q$. Thus, $(D, \uparrow) \in \mathsf{div}(D_2, Q)$.

In fact, $(D, \uparrow) \in \mathsf{mdiv}(D_2, Q)$. To see why that is the case, we observe that every database D_3 for which $D_3 \preceq D$, but $D \not\preceq D_3$, is isomorphic to some database obtained from D by dropping facts or renaming non-isolated constants – renaming isolated constants would simply result into an equivalent database. It can be checked that for any such database D_3, the addition of guarded unravelings of D_2 is no longer enough to entail Q. Consider for example the database $D_3 = \{R(a, b_1), W(d_1, b_2, c), U(c, d_2, a)\}$ obtained from D by renaming the two occcurrences of b as b_1 and b_2. Also, let $\downarrow$ be the homomorphism from D_3 to D_2 which is the identity on a and c and maps b_1 and b_2 to b and d_1 and d_2 to d. Then, $\mathsf{ext}(D_3, \downarrow, D_2) \not\models Q$: while $\mathsf{ch}_{\mathcal{O}}(\mathsf{ext}(D_3, \downarrow, D_2))$ will contain atoms over R, S, and T (as requested by q), the underlying hypergraph structure of D needed to entail q is lost.

On the other hand, q maps into $\mathsf{ch}_{\mathcal{O}}(D^+)$ via a homomorphism h which maps x, y, z to a, b, c. The hypergraph H_q is isomorphic to a sub-hypergraph of H_D. In this sense, H_D preserves the structure of D_2 needed to entail q. At the same time, the atoms added by guarded unravelings, like $V(c, a)$, allow the entailment of particular atoms from q, like $T(x, z)$.

We next show how given a homomorphism h from a CQ p in an OMQ Q to the chase of a database of the form $\mathsf{ext}(D, \uparrow, D_0)$, it is possible to construct a diversification of D, $(D', \downarrow)$, such that the database obtained by extending D' with guarded unravelings of D_0 according to the composition homomorphism $\uparrow \circ \downarrow$ from D' still entails the OMQ Q.

▶ **Lemma 10.** *Let $Q = (\mathcal{O}, \mathbf{S}, q)$ be an OMQ from (GDLog, UCQ), D and D_0 be $\mathbf{S}$-databases, and $\uparrow$ a homomorphism from D to D_0 which is i.g.s. such that $\mathsf{ext}(D, \uparrow, D_0) \models Q$. Also, let h be a homomorphism from some CQ p in q to $\mathsf{ch}_{\mathcal{O}}(\mathsf{ext}(D, \uparrow, D_0))$. Then, there exists a diversification $(D', \downarrow)$ of D such that:*

1. *$\downarrow$ is the identity function on $\mathsf{ker}(D')$;*
2. *$\mathsf{ker}(D') \subseteq \mathsf{ran}(h) \cap \mathsf{dom}(D)$;*
3. *$\mathsf{ext}(D', \uparrow \circ \downarrow, D_0) \models Q$.*

Proof. We start by partitioning the set of variables from the CQ p, $\mathsf{dom}(p)$, into maximal sets $A_0, A_1, \ldots A_n$, such that:

1. $h(A_0) \subseteq \mathsf{dom}(D)$;
2. for every $i > 0$, $h(A_i) \subseteq \mathsf{dom}(D_0^{\mathbf{a}_i}) \setminus \mathsf{dom}(D)$, where $\mathbf{a}_i$ is some maximal guarded set in D and $D^{\mathbf{a}_i}$ is the corresponding guarded unraveling of D_0 which has been added to D during the construction of $\mathsf{ext}(D, \uparrow, D_0)$.

Thus, A_0 is the set of all variables which map into $\mathsf{dom}(D)$, and each A_i is a maximal set of variables which map into constants from some guarded unraveling of D_0 which are not from D. Note that by maximality of the sets A_i, it follows that $\mathbf{a}_i \neq \mathbf{a}_j$, whenever $i \neq j$.

We now proceed to defining D'. We start by initializing it as the set of atoms $\{r(\mathbf{c}) \in D \mid \mathbf{c} \subseteq \mathsf{ran}(h)\}$. These are exactly the atoms for which all their terms are from $h(A_0)$. Then, for every i, with $1 \leq i \leq n$, there must be some atom $r(\mathbf{a}_i)$ in D (where $\mathbf{a}_i$ is as above in the definition of the sets A_i). Note that there might be more than one such atom, however it is enough to select one. We rename all constants from $\mathbf{a}_i$ which are not from $\mathsf{ran}(h)$ as fresh constants and add the atom to D'. Intuitively, each such atom will serve as a skeleton, as a guard for adding guarded unravelings of D_0 to D'.

By construction, $\mathsf{dom}(D) \cap \mathsf{ran}(h) \subseteq \mathsf{dom}(D')$. All other constants from $\mathsf{dom}(D')$ are fresh, and thus isolated. Thus, $\mathsf{ker}(D') \subseteq \mathsf{dom}(D) \cap \mathsf{ran}(h)$ (Point (2) of the Lemma). We define $\downarrow$ as a mapping from $\mathsf{dom}(D')$ to $\mathsf{dom}(D)$ which is the identity on $\mathsf{ran}(h) \cap \mathsf{dom}(D)$ and which maps fresh constants to original constants for the remaining elements of $\mathsf{dom}(D')$. It can be verified easily that $(D', \downarrow)$ is a diversification of D and that $\downarrow$ is the identity on $\mathsf{ker}(D')$ – Point (1) of the Lemma. It remains to show Point (3).

Let $g = \uparrow \circ \downarrow$ and $D^+ = \mathsf{ext}(D', g, D_0)$. For every i with $0 < i \leq n$, we define an isomorphism κ_i from the database $D_0^{\mathbf{a}_i}$, the guarded unraveling of D_0 added to the maximal guarded set $\mathbf{a}_i$ during the construction of $\mathsf{ext}(D, \uparrow, D_0)$ to the database $D_0^{\mathbf{b}_i}$, the copy of $D_0^{\mathbf{a}_i}$ added to D' during the construction of D^+, in which constants from $\mathbf{a}_i$ have been replaced with constants from $\mathbf{b}_i$. Note that $\kappa_i(\mathbf{a}_i) = \mathbf{b}_i$ and $\downarrow(\mathbf{b}_i) = \mathbf{a}_i$.

We construct a mapping h' from $\mathsf{dom}(p)$ to $\mathsf{dom}(D^+)$ as follows:

$$h'(x) = \begin{cases} h(x) & \text{if } x \in A_0 \\ \kappa_i(h(x)) & \text{if } x \in A_i \end{cases}$$

Due to the property of guarded unravelings to preserve atomic consequences, it can be shown that for every i with $0 < i \leq n$, κ_i is an isomorphism also from the restriction of $\mathsf{ch}_{\mathcal{O}}(\mathsf{ext}(D, \uparrow, D_0))$ to $\mathsf{dom}(D_0^{\mathbf{a}_i})$ to the restriction of $\mathsf{ch}_{\mathcal{O}}(D^+)$ to $\mathsf{dom}(D_0^{\mathbf{b}_i})$. Furthermore, again due to the property of guarded unravelings to preserve atomic consequences, it can be shown that for every atom $r(\mathbf{c})$ in $\mathsf{ch}_{\mathcal{O}}(\mathsf{ext}(D, \uparrow, D_0))$ for which $\mathbf{c} \subseteq \mathsf{ran}(h)$, it is the case that $r(\mathbf{c})$ in $\mathsf{ch}_{\mathcal{O}}(D^+)$ as well. Then it can be shown that h' is a homomorphism from p to $\mathsf{ch}_{\mathcal{O}}(D^+)$. ◀

Lemma 10 will be useful later as it offers a way to construct databases based on diversifications which are progressively smaller w.r.t. the homomorphism order.

3.4 Characteristic Databases and Covers

In this section we put together the notions introduced in previous subsections to define *(extended) characteristic databases* and *covers* of an OMQ.

▶ **Definition 11.** *For Q an OMQ from (GDLog, UCQ),* the set of characteristic databases for Q *is* $\mathbb{D}_Q = \{D \mid (D, \uparrow) \in \mathsf{mdiv}(D_0, Q), D_0$ *is qi w.r.t.* $Q\}$. The set of extended characteristic databases for Q *is* $\mathbb{D}_Q^+ = \{\mathsf{ext}(D, \uparrow, D_0) \mid (D, \uparrow) \in \mathsf{mdiv}(D_0, Q), D_0$ *is qi w.r.t.* $Q\}$.

Thus, a characteristic database D is part of a minimal diversification $(D, \uparrow)$ w.r.t. Q of a database D_0 which is qi w.r.t. Q. It is easy to see that:

▶ **Lemma 12.** *For Q an $\mathbf{S}$-OMQ from (GDLog, UCQ) and D an $\mathbf{S}$-database such that $D \models Q$, there exists a database $D' \in \mathbb{D}_Q^+$ such that $D' \to D$.*

Furthermore, characteristic databases have the following properties:

▶ **Lemma 13.** *Let $Q = (\mathcal{O}, \mathbf{S}, q)$ be an OMQ from (GDLog, UCQ), $\mathsf{ext}(D, \uparrow, D_0)$ be a database from $\mathbb{D}_Q^+$, and h be a homomorphism from a CQ p in q to $\mathsf{ch}_{\mathcal{O}}(\mathsf{ext}(D, \uparrow, D_0))$. Then:*

1. $\mathsf{ker}(D) \subseteq \mathsf{ran}(h)$*;*
2. *there exists a computable function f such that $|D| \leq f(|Q|)$.*

Proof. We start by showing Point (1). From Lemma 10 we know that there exists a database D' and a homomorphism $\downarrow$ from D' to D such that $(D', \downarrow)$ is a diversification of D. Thus, $D' \preceq D$. As $(D, \uparrow)$ is a diversification of D_0, it follows that $(D', \uparrow \circ \downarrow)$ is a diversification of D_0. From Point (3) of Lemma 10 we obtain that $(D', \uparrow \circ \downarrow)$ is a diversification of D_0 w.r.t. Q. In other words, $D' \in \mathsf{div}(D_0, Q)$. As $D \in \mathsf{mdiv}(D_0, Q)$, and $D' \preceq D$, it must be the case that $D \preceq D'$ (otherwise, D would not be minimal). Thus, $D \leftrightarrow D'$.

From Point (1) of Lemma 10, we know that D' maps into D via the function $\downarrow$ which is the identity on $\mathsf{ker}(D')$. Thus, $\mathsf{ker}(D') \subseteq \mathsf{ker}(D)$ (a non-isolated constant, i.e. a constant which occurs in $\mathsf{ker}(D')$ can only map into another non-isolated constant, thus a constant

from $\mathsf{ker}(D)$). As D is a core, it is the case that D maps into D' only via injective homomorphisms: otherwise, the composition of a non-injective homomorphism from D to D' with the homomorphism $\downarrow$ from D' to D would lead to a non-injective endomorphism on D which is in contradiction to the fact that D is a core. As every homomorphism from D to D' maps $\mathsf{ker}(D)$ into $\mathsf{ker}(D')$ injectively, it is the case that $|\mathsf{ker}(D)| \leq |\mathsf{ker}(D')|$. Thus, $\mathsf{ker}(D') = \mathsf{ker}(D)$. Finally, from Point (2) of Lemma 10 we know that $\mathsf{ker}(\mathsf{D}') \subseteq \mathsf{ran}(h) \cap \mathsf{dom}(D)$, and thus $\mathsf{ker}(D) \subseteq \mathsf{ran}(h)$.

Point (2) follows from Point (1) and from the fact that D is a core: as $|\mathsf{ker}(D)|$ is bounded in $|Q|$, $|D|$ will be bounded in $|Q|$ as well. ◀

We next define the *cover* $Q_c = (\mathcal{O}, \mathbf{S}, q_c)$ of an OMQ $Q = (\mathcal{O}, \mathbf{S}, q)$ by using the set of extended characteristic databases for Q, $\mathbb{D}^+_Q$. To construct CQs in q_c, we will consider images of CQs from q in the chase of some database D^+ from $\mathbb{D}^+_Q$ together with extra atoms (facts)[1] from D^+ which will guard atoms in the CQ image. We want to include guards from D^+ which cover as many atoms as possible from the image of the CQ, and thus to potentially decrease submodular width as in Example 1. Formally:

▶ **Definition 14.** *For $Q = (\mathcal{O}, \mathbf{S}, q)$ an OMQ from (GDLog, UCQ), its* cover *is the OMQ $Q_c = (\mathcal{O}, \mathbf{S}, q_c)$, with q_c the union of all CQs p_c with $D[p_c]$ of the form $h(p) \cup S$, where:*

1. *h is a homomorphism from some CQ p in q_c to $\mathsf{ch}_{\mathcal{O}}(D^+)$, where D^+ is a database of the form $\mathsf{ext}(D, \uparrow, D_0)$ from $\mathbb{D}^+_Q$;*
2. *S is a minimal set of atoms such that:*
   a. *$D \subseteq S \subseteq D^+$;*
   b. *for every atom $r(\mathbf{a})$ from $h(p)$, there exists an atom $r'(\mathbf{a}')$ from S such that $\mathbf{a} \subseteq \mathbf{a}'$ and $\mathbf{a}'$ is a maximal guarded set in D^+.*

The set of atoms S in Definition 14 is the set of guards added to the image $h(p)$ of a CQ p in q. By including D, S trivially guards every maximal set of atoms S' from $h(p)$ for which $\mathsf{dom}(S')$ is a guarded set in D. All other atoms from $h(p)$ will map into the guarded unraveling part of (the chase of) D^+. For every maximal set S' of such atoms for which $\mathsf{dom}(S')$ is a guarded set in D^+, there exists a unique atom $r'(\mathbf{a}')$ in D^+ such that $\mathsf{dom}(S') \subseteq \mathbf{a}'$. Thus, the definition insures that "maximal" guards, which cover as many atoms as possible, are added to $h(p)$.

▶ **Example 15.** Let $\mathbb{Q}$ be as in Example 1. For every $i > 1$, let $D_{i,0}$ be the $\mathbf{S}_i$-database: $\{S_i(\mathbf{a}_i), T(a_1, a_2), T(a_1, a_3), \ldots T(a_{i-1}, a_i)\}$. We have that $D_{i,0} \models Q_i$ and furthermore, $D_{i,0}$ is qi w.r.t. Q_i: q_i is the only contraction mapping into $\mathsf{ch}_{\mathcal{O}_i}(D_{i,0})$. Let $D_i = \{S(\mathbf{a}_i)\}$. Then $(D_i, \uparrow) \in \mathsf{mdiv}(D_{i,0}, Q)$, where $\uparrow$ is the identity function on $\mathbf{a}_i$: any guarded unraveling of $D_{i,0}$ at $\mathbf{a}_i$ will add back all the facts from $D_{i,0}$. Thus, $D_i \in \mathbb{D}_{Q_i}$ and for every database of the form $\mathsf{ext}(D_i, \uparrow, D_{i,0})$, $D_{i,0} \subseteq \mathsf{ext}(D_i, \uparrow, D_{i,0})$ (there might be more facts in $\mathsf{ext}(D_i, \uparrow, D_{i,0})$). In the following we fix such a database $\mathsf{ext}(D_i, \uparrow, D_{i,0})$ and refer to it as D^+_i.

The CQ q_i maps then into $\mathsf{ch}_{\mathcal{O}_i}(D^+_i)$ via a homomorphism h with range $\mathbf{a}_i$. We construct a CQ p_c belonging to the cover $Q_{i,c}$ of Q_i based on h and D^+_i: $D[p_c]$ will be the union of $\psi^R_i(\mathbf{a}_i) \wedge \psi^T_i(\mathbf{a}_i)$, the image of q_i under h, with the singleton set $\{S(\mathbf{a}_i)\}$ from D^+_i: $D[p_c] = \psi^R_i(\mathbf{a}_i) \wedge \psi^T_i(\mathbf{a}_i) \wedge S(\mathbf{a}_i)$. We note that $S(\mathbf{a}_i)$ is a guard for all atoms from $h(q_i)$ and $D[p_c]$ is isomorphic to $D[q'_i]$ from Example 1. It can be shown that p_c is the unique such CQ (up to isomorphism) and thus $Q_{i,c}$ is identical to the OMQ Q'_i from Example 1.

[1] As we construct a CQ by conjoining the image of an original CQ into a database with more facts from the database, the distinction between facts and atoms is blurred.

As the next lemma shows, covers are equivalent to original OMQs. Intuitively, this is the case as we only add atoms to CQs in the original OMQ from databases which entail the OMQ and which are small w.r.t. the homomorphism order:

▶ **Lemma 16.** *For $Q = (\mathcal{O}, \mathbf{S}, q)$ an OMQ from (GDLog, UCQ), and Q_c its cover, $Q \equiv Q_c$.*

Proof. It is clear that $Q_c \subseteq Q$. We show that $Q \subseteq Q_c$. For D an $\mathbf{S}$-database such that $D \models Q$, there exists a database $D_0 \to D$ which is qi w.r.t. Q. Furthermore, there exists a diversification $(D_1, \uparrow) \in \mathsf{mdiv}(D_0, Q)$. Thus, $\mathsf{ext}(D_1, \uparrow, D_0) \in \mathbb{D}^+_Q$, and there exists some CQ p in q s.t. p maps into $\mathsf{ch}_\mathcal{O}(\mathsf{ext}(D_1, \uparrow, D_0))$ via some homomorphism h. We construct a CQ p_c from q_c as in Definition 14 based on p, h, and $\mathsf{ch}_\mathcal{O}(\mathsf{ext}(D_1, \uparrow, D_0))$. Then, $D[p_c] \subseteq \mathsf{ch}_\mathcal{O}(\mathsf{ext}(D_1, \uparrow, D_0))$, thus $p_c \to \mathsf{ch}_\mathcal{O}(\mathsf{ext}(D_1, \uparrow, D_0))$, and as p_c is a CQ in q_c, it follows that $\mathsf{ext}(D_1, \uparrow, D_0) \models Q_c$. As $\mathsf{ext}(D_1, \uparrow, D_0) \to D_0 \to D$, it follows that $D \models Q_c$. ◀

Thus covers are a good candidate for witnesses of low semantic SMW for OMQs from (GDLog, UCQ). We defer to the next section the result detailing in which way they serve as such witnesses. For now we show that they have finite bounded size and are computable:

▶ **Lemma 17.** *For $Q = (\mathcal{O}, \mathbf{S}, q)$ an OMQ from (GDLog, UCQ) and $Q_c = (\mathcal{O}, \mathbf{S}, q_c)$ its cover, there exists a computable function g such that for every CQ p_c in q_c, $|p_c| \leq g(|Q|)$.*

Proof. Each p_c has the form $h(p) \cup S$, where p is some CQ in q, h a homomorphism from p to $\mathsf{ch}_\mathcal{O}(\mathsf{ext}(D, \uparrow, D_0))$, for some $\mathsf{ext}(D, \uparrow, D_0) \in \mathbb{D}^+(Q)$ and S the extension of D with atoms from $\mathsf{ext}(D, \uparrow, D_0) \in \mathbb{D}^+(Q)$ which guard atoms in $h(p)$. As $|h(p)|$ is bounded in $|Q|$, $|D| \leq f(|Q|)$, for some computable function f (from Lemma 13) and S contains at most one atom for each atom in $h(p)$, it follows that $|p_c|$ is bounded in $|Q|$. ◀

▶ **Lemma 18.** *For every OMQ $Q = (\mathcal{O}, \mathbf{S}, q) \in (GDLog, UCQ)$, $Q_c = (\mathcal{O}, \mathbf{S}, q_c)$ is computable.*

Proof Sketch. Let $\mathbf{S}'$ be the extension of $\mathbf{S}$ with relational symbols which occur in Q. Also, let *the diameter of* Q be the maximum between the arity of some relational symbol from $\mathbf{S}$ and the number of variables in some CQ in q. We devise an encoding into Guarded Second Order Logic (GSO) [23] to check whether some structure over the extended signature $\mathbf{S}'$ of size bounded by $g(|Q|)$ (with g as in Lemma 17) is of the right form to be part of q_c. While GSO is in general undecidable, on sparse structures it has the same expressivity as MSO [9] and thus it is decidable. For our purposes, it is possible to restrict to structures of treewidth bounded by the diameter of the given OMQ, which are sparse [18]. ◀

4 Main Results

In this section we establish the main results of our paper: a characterization for fixed-parameter tractability of OMQs from (GDLog, UCQ), a reduction from parameterized uniform CSPs to parameterized OMQs from (GDLog, UCQ), and a semantic characterization for fixed-parameter tractability of OMQs from (GTGD, UCQ). We conclude the section with a discussion concerning covers of OMQs from (GDLog, UCQ): we show that they are indeed witnesses for bounded semantic submodular width as anticipated in the previous section.

4.1 Results for (GDLog, UCQ)

For a class $\mathbb{Q}$ of OMQs from (GDLog, UCQ), we denote with $\mathbb{D}_\mathbb{Q}$ the class of characteristic databases for OMQs from $\mathbb{Q}$ and with $\mathbb{Q}_c$ the class of covers for OMQs from $\mathbb{Q}$. The first main result which we show in this section is as follows:

▶ **Theorem 19** (Main Result 3). *Let $\mathbb{Q}$ be a r.e. enumerable class of OMQs from (GDLog, UCQ). Under the Exponential Time Hypothesis, the following are equivalent:*

1. *p-OMQ($\mathbb{Q}$) is fixed-parameter tractable*
2. *$\mathbb{Q}_c$ has bounded sub-modular width*
3. *$\mathbb{D}_\mathbb{Q}$ has bounded sub-modular width.*

Towards showing the result, the following fpt-reduction from evaluation of parameterized CSP to evaluation of parameterized OMQs from (GDLog, UCQ) plays an important role.

▶ **Theorem 20** (Main Result 2). *Let $\mathbb{Q}$ be a r.e. enumerable class of OMQs from (GDLog, UCQ). Then, there exists an fpt-reduction from p-CSP($\mathbb{D}_\mathbb{Q}$, _) to p-OMQ($\mathbb{Q}$).*

Proof. Let (D, B) be an instance of p-CSP($\mathbb{D}_\mathbb{Q}$, _). Also, let π be the projection mapping from $D \times B$ to D and let D_2 be the database obtained from $D \times B$ by dropping all atoms $R(\mathbf{a})$ for which $\pi|_\mathbf{a}$ is not injective. Then, $\pi|_{\mathsf{dom}(D_2)}$ is a homomorphism from D_2 to D which is i.g.s. As $D \in \mathbb{D}_\mathbb{Q}$, there exists an OMQ $Q \in \mathbb{Q}$, such that $D \in \mathbb{D}_Q$. Thus, there must some database D_0 which is qi w.r.t. Q, and a homomorphism $\uparrow$ from D to D_0 such that $(D, \uparrow) \in \mathsf{mdiv}(D_0, Q)$. Then $\pi \circ \uparrow$ is a homomorphism from D_2 to D_0 which is i.g.s. We can find Q and D_0 by enumeration, as all necessary tests can be decided using GSO encodings, similarly to the proof of Lemma 18. In the following, we denote with D^+ and D_2^+, the $\mathbf{S}$-databases $\mathsf{ext}(D, \uparrow, D_0) \in \mathbb{D}_Q^+$ and $\mathsf{ext}(D_2, \pi \circ \uparrow, D_0)$, resp. We will show that $D \to B$ iff $D_2^+ \models Q$. As $|D_2^+|$ is linear in $|B|$, we have an fpt-reduction.

'$\Rightarrow$': Assume that $D_2^+ \models Q$. The strategy of the proof is as follows: using Lemma 10 we construct a diversification (D_3, ι) of D_2 and then using the qi property of D_0, and subsequently of D^+, we show that $D_3 \preceq D$ and that $(D_3, g) \in \mathsf{div}(D_0, Q)$, for some homomorphism g. But $(D, \uparrow)$ is a minimal diversification of D_0 w.r.t. Q. Thus, $D \preceq D_3$, so $D \to D_3 \to D_2 \to D \times B \to B$.

Now to the details. Let p' be some contraction of a CQ p in Q which maps injectively only via an injective homomorphism h' into $\mathsf{ch}_\mathcal{O}(D_2^+)$ (denoted $p' \to^{io} \mathsf{ch}_\mathcal{O}(D_2^+)$) and let $A = \mathsf{ran}(h') \cap \mathsf{dom}(D_2)$. Also, let π^+ be the extension of the projection homomorphism π from D_2 to D to a homomorphism from D_2^+ to D^+. Then, π^+ is a homomorphism also from $\mathsf{ch}_\mathcal{O}(D_2^+)$ to $\mathsf{ch}_\mathcal{O}(D^+)$ and $\pi^+ \circ h'$ is a homomorphism from p' to $\mathsf{ch}_\mathcal{O}(D^+)$. As D_0 is qi w.r.t Q, according to Lemma 4, so is D^+. Also, as $p' \to^{io} \mathsf{ch}_\mathcal{O}(D_2^+)$, and $D_2^+ \to D^+$, it must be the case that p' maps injectively ony into $\mathsf{ch}_\mathcal{O}(D^+)$ ($p' \to^{io} \mathsf{ch}_\mathcal{O}(D^+)$). Otherwise, there would be some contraction p'' of p' such that $p'' \to \mathsf{ch}_\mathcal{O}(D^+)$ and $p'' \not\to \mathsf{ch}_\mathcal{O}(D_2^+)$ which would violate the fact that D^+ is qi w.r.t. Q. Thus, $\pi^+ \circ h'$ is injective.

Further on, as $A \subseteq \mathsf{ran}(h')$, it follows that $\pi^+_{|A}$ is injective. Also, as $A \subseteq \mathsf{dom}(D_2)$, it follows that $\pi_{|A}$ is injective.

As D_2^+ is of the form $\mathsf{ext}(D_2, \pi \circ \uparrow, D_0)$, according to Lemma 10 there exists a diversification (D_3, ι) of D_2 such that ι is the identity on $\mathsf{ker}(D_3)$, $\mathsf{ker}(\mathsf{D_3}) \subseteq A$, and $\mathsf{ext}(D_3, g, D_0) \models Q$, where g is the composition homomorphism $\uparrow \circ \pi \circ \iota$ from D_3 to D_0. Let D_3^+ be $\mathsf{ext}(D_3, g, D_0)$. Note that g is i.g.s. Also, let $h = \pi \circ \iota$ be the composition homomorphism from D_3 to D. It can be seen that h is also i.g.s. Figure 1 provides an overview of all the homomorphisms employed in the proof: the markings "io" on arrows denote the existence of injective only homomorphisms. Note that while the figure depicts extended databases, e.g. databases like $\mathsf{ext}(D, \uparrow, D_0)$ (D^+), most of the depicted homomorphisms are between the "base" databases, e.g. D_3, D_2, D, D_0.

We show next that $h|_{\mathsf{ker}(D_3)}$ is also injective. We have that $\mathsf{ran}(\iota_{|\mathsf{ker}(\mathsf{D_3})}) = \mathsf{ker}(\mathsf{D_3})$, thus $h|_{\mathsf{ker}(D_3)} = \pi \circ \iota_{|\mathsf{ker}(\mathsf{D_3})}$ is the same as $\pi_{|\mathsf{ker}(\mathsf{D_3})} \circ \iota_{|\mathsf{ker}(\mathsf{D_3})}$. As $\mathsf{ker}(\mathsf{D_3}) \subseteq A$, $\pi_{|A}$ is injective, and $\iota_{|\mathsf{ker}(\mathsf{D_3})}$ is the identity function, it follows that $h|_{\mathsf{ker}(D_3)}$ is injective. Recall that h is also i.g.s.

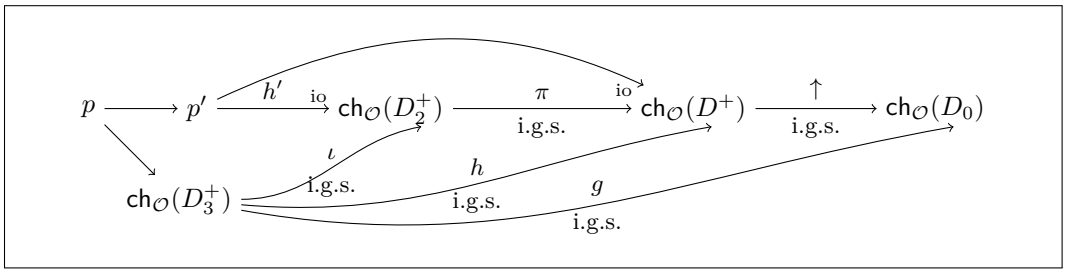


Figure 1 Constructions for Proof of Direction ⇒ of Theorem 20.

Thus, (D_3, h) is a diversification of D and $D_3 \preceq D$. As $D \preceq D_0$, it follows that $D_3 \preceq D_0$. At the same time, $\mathsf{ext}(D_3, g, D_0) \models Q$, thus $(D_3, g) \in \mathsf{div}D_0, Q$. But $(D, \uparrow) \in \mathsf{mdiv}(D_0, Q)$, thus $D \preceq D_3$. Thus, $D \to D_3 \to D_2 \to D \times B \to B$.

'⇐': Assume that $D \to B$. Then, D maps into $D \times B$ via some homomorphism h. Let $A = \mathsf{ran}(h)$. At the same time, $D \times B$ maps into D via the projection mapping π. Thus, $\pi \circ h$ is a homomorphism from D to itself. As D is a core, $\pi \circ h$ must be injective, and thus $\pi|_A$ is injective. Then, the database $(D \times B)|_A$ is a sub-structure of D_2, the structure obtained from $D \times B$ by removing all facts $r(\mathbf{a})$ for which $\pi|_{\mathbf{a}}$ is not injective. Thus, h is a homomorphism from D to D_2, or $D^+ \to D_2^+$. As $D^+ \models Q$, it follows that $D_2^+ \models Q$. ◀

We now prove the counter-positive of direction "1 ⇒ 3" of Theorem 19. Let us assume that $\mathbb{D}_\mathbb{Q}$ has unbounded submodular width. As all $D \in \mathbb{D}_\mathbb{Q}$ are cores and cores witness semantic submodular width [16], $\mathbb{D}_\mathbb{Q}$ must have unbounded semantic submodular width. As explained in the Introduction, the following is known about evaluating uniform CSPs:

▶ **Theorem 21** (Theorem 1, [16]). *Let $\mathbb{C}$ be a recursively enumerable class of structures. Assuming the Exponential Time Hypothesis, p-CSP($\mathbb{C}$, _) is fixed-parameter tractable if and only if $\mathbb{C}$ has bounded semantic submodular width.*

Then, according to Theorem 21, p-CSP($\mathbb{D}_\mathbb{Q}$, _) is not fpt, and based on our reduction from Theorem 20, p-OMQ($\mathbb{Q}$) is not fpt either.

To establish direction "3 ⇒ 2" of Theorem 19, we show the following:

▶ **Lemma 22.** *Let $Q = (\mathcal{O}, \mathbf{S}, q)$ be an OMQ from (GDLog, UCQ) and $Q_c = (\mathcal{O}, \mathbf{S}, q_c)$ be its cover. Then, $\mathsf{SMW}(Q_c) \leq \mathsf{SMW}(\mathbb{D}_Q)$.*

Proof Sketch. We show that for every CQ p_c in q_c there exists a database in $\mathbb{D}_Q$ of higher or equal submodular width. Every such p_c is of the form $h(p) \cup S$, where h is a homomorphism from a CQ p in q to $\mathsf{ch}_\mathcal{O}(D^+)$ for some $D^+ \in \mathbb{D}_Q^+$ and S is a set of atoms such that $D \subseteq S \subseteq D^+$ and every atom $r(\mathbf{a})$ from $S \setminus D$ has the property that $\mathbf{a}$ is a maximal guarded set in D^+. The latter follows from the minimality requirement on S in Definition 14 as well as from Point (2b) of the same definition. As every atom from $h(p)$ is guarded by some atom in S, it follows that $\mathsf{SMW}(p_c) = \mathsf{SMW}(S)$. Then, by careful manipulations of tree decompositions on D, D^+, and S, it can be further shown that: $\mathsf{SMW}(S) \leq \mathsf{SMW}(D)$. ◀

We next show the upper bound, i.e. direction "2 ⇒ 1" of Theorem 19: if $\mathbb{Q}_c$ has bounded SMW, then, p-OMQ($\mathbb{Q}$) is fixed-parameter tractable. The result follows from the fact that for an OMQ Q, Q_c is computable from Q (Lemma 18) and the following lemma (which concerns also OMQs based on GTGD):

▶ **Lemma 23.** *Let Q be an OMQs from (GTGD, UCQ) of bounded submodular width. Then, p-OMQ(Q) is fixed-parameter tractable.*

Proof. We use some results from [3]. Let L be the language of *linear tgds*, i.e. tgds which contain only one atom in the body. For an OMQ $Q = (\mathcal{O}, \mathbf{S}, q)$ from (GTGD, UCQ), and a database D, Lemma A.3 from [3], shows how to construct in FPT another OMQ $(\mathcal{O}^*, \mathbf{S}^*, q)$ from (L, UCQ) and a database D^* such that $D \models Q$ iff $D^* \models Q^*$. Then, Lemma A.1 from [3] shows that deciding whether $D^* \models Q^*$ can be done by considering a finite portion of $\mathsf{ch}_{\mathcal{O}^*}(D^*)$, which again can be computed in FPT. Thus, $D \models Q$ iff $D' \models q$, where D' is a database which can be computed in FPT. Thus, assuming that q has bounded submodular width, according to Theorem 21, deciding whether $D' \models q$ is in FPT as well. ◀

4.2 Results for (GTGD, UCQ)

The main result which we show in this section is as follows.

▶ **Theorem 24** (Main Result 1)**.** *Let $\mathbb{Q}$ be a r.e. class of OMQs from (GTGD, UCQ). Under the Exponential Time Hypothesis, $\mathbb{Q}$ has bounded semantic submodular width iff p-OMQ($\mathbb{Q}$) is fixed-parameter tractable.*

To show the result, we exploit the fact that for every OMQ $Q \in (\text{GTGD}, \text{UCQ})$, there exists an OMQ $Q' \in (\text{GDLog}, \text{UCQ})$, *the existential rewriting of* Q, such that $Q \equiv Q'$ [3]. Let $\mathbb{Q}'$ be the corresponding class of existential rewritings of OMQs from $\mathbb{Q}$.

Then, if p-OMQ($\mathbb{Q}$) is fpt, so is p-OMQ($\mathbb{Q}'$) and according to Theorem 19, the class of covers of OMQs from $\mathbb{Q}'$, $\mathbb{Q}'_c$, has bounded submodular width. But, $\mathbb{Q} \equiv \mathbb{Q}'_c$, and thus, $\mathbb{Q}$ has bounded semantic submodular width.

We turn our attention to the other direction of the theorem. In this case there exists a class of OMQs $\mathbb{Q}_k$ from (GTGD, UCQ) of bounded submodular width, such that $\mathbb{Q} \equiv \mathbb{Q}_k$. According to Lemma 23, p-OMQ($\mathbb{Q}_k$) is fpt, but we do not know how to compute $\mathbb{Q}_k$. The same holds about the class of its existential rewritings $\mathbb{Q}'_k$ from (GDLog, UCQ): it can be evaluated in fpt, but it is not given a priori. From Theorem 19 we know that the set of characteristic databases of $\mathbb{Q}'_k$, $\mathbb{D}_{\mathbb{Q}'_k}$ must have bounded submodular width.

A natural question is whether the notions used for characterizing OMQs from (GDLog, UCQ) are semantic, e.g. do equivalent OMQs have the same set of characteristic databases? If that would be the case, as $\mathbb{Q}' \equiv \mathbb{Q}'_k$, the set of characteristic databases $\mathbb{D}_{\mathbb{Q}'}$ would be identical to $\mathbb{D}_{\mathbb{Q}'_k}$, and thus would have bounded submodular width. Then, according to Theorem 19, p-OMQ($\mathbb{Q}'$) is fpt and so is p-OMQ($\mathbb{Q}$). However, this is not the case:

▶ **Example 25.** Let $Q_1 = (\mathcal{O}_1, \mathbf{S}, q_1)$ and $Q_2 = (\mathcal{O}_2, \mathbf{S}, q_2)$ be the following two OMQs:

$$\begin{array}{lll} \mathcal{O}_1 = \{R(x,y) \to A(x)\} & \mathbf{S} = \{R\} & q_1 = \exists x\ A(x) \\ \mathcal{O}_2 = \emptyset & \mathbf{S} = \{R\} & q_2 = \exists x, y\ R(x,y) \end{array}$$

Also, let $D = \{R(a,a)\}$ be an $\mathbf{S}$-database. It can be checked that $Q_1 \equiv Q_2$, $D \models Q_1$, and $D \models Q_2$. However, D is qi w.r.t. Q_1, but not w.r.t. Q_2 and $D \in \mathbb{D}_{Q_1}$, but $D \notin \mathbb{D}_{Q_2}$.

Still, when considering two equivalent OMQs Q_1 and Q_2 from (GDLog, UCQ), it is possible to construct another equivalent OMQ Q_{12} based on the intersection of the sets of (extended) characteristic databases of the two OMQs, $\mathbb{D}_{Q_1}$ and $\mathbb{D}_{Q_2}$. This can be done by starting with Q_1 (or Q_2 for that matter) and following the process described in Section 3 to construct the cover $Q_{1,c}$ of Q_1, except that in this case we use $\mathbb{D}_{Q_1} \cap \mathbb{D}_{Q_2}$ instead of $\mathbb{D}_{Q_1}$.

The new OMQ $Q_{12} = (\mathcal{O}, \mathbf{S}, q_{12})$ has the property that every CQ in q_{12} is a CQ in $q_{1,c}$, the UCQ from the cover $Q_{1,c}$ of Q_1. Thus $Q_{12} \subseteq Q_{1,c}$. At the same time, $Q_1 \subseteq Q_{12}$, and thus $Q_1 \equiv Q_{12}$. The submodular width of q_{12} will be bounded by the submodular width of $\mathbb{D}_{Q_1} \cap \mathbb{D}_{Q_2}$. Thus, if the submodular width of $\mathbb{D}_{Q_2}$ is bounded by some $k > 0$, there exists an OMQ $Q_{12} \equiv Q_1$ of submodular width bounded by k which can be constructed by removing all CQs with submodular width greater than k from the cover of Q_1. This allows us to prove direction "⇒" of Theorem 24: given the existence of (GDLog, UCQ) fpt witnesses $\mathbb{Q}'_k$ as above, we can construct from $\mathbb{Q}'$ a new class of (GDLog, UCQ) OMQs $\mathbb{Q}''$ of bounded submodular width which is equivalent to $\mathbb{Q}'$ and thus also to $\mathbb{Q}$. Thus, p-OMQ($\mathbb{Q}$) is fpt.

Finally, we obtain as a corollary of Theorem 19 and Theorem 24 the following result concerning covers which was already anticipated in Section 3:

▶ **Corollary 26.** *Let $\mathbb{Q}$ be a class of OMQs from (GDLog, UCQ). Then, $\mathbb{Q}$ has bounded semantic submodular width iff its class of covers $\mathbb{Q}_c$ has bounded submodular width.*

The question remains open whether covers are witnesses for semantic submodular width when considered individually, as opposed to witnesses for bounded semantic submodular width when considered as classes, as established in Corollary 26. We leave this open for now.

5 Revisiting the Bounded Arity Case

We revisit the characterization for fpt evaluation of OMQs from (GTGD, UCQ) over bounded arity schemas from [3]. A class of OMQs $\mathbb{Q}$ is *over bounded arity schemas* if there exists r such that every schema in some OMQ in $\mathbb{Q}$ contains only symbols of arity at most r.

▶ **Theorem 27** (Theorem 5.3, [3]). *Let $\mathbb{Q}$ be a r.e. class of OMQs from (GTGD, UCQ) over bounded arity schemas. Assumming* FPT $\neq$ W[1]*, the following are equivalent :*
1. *p-OMQ($\mathbb{Q}$) is fixed-paramater tractable.*
2. *$\mathbb{Q}$ has bounded semantic treewidth.*

If either statement is false, then p-OMQ($\mathbb{Q}$) is W[1]*-hard.*

The characterization generalizes a previous result concerning complexity of evaluating OMQs from ($\mathcal{ELHI}_\bot$, UCQ) [4]. At the same time, it can be seen as a generalization of Grohe's complexity results regarding the parameterized complexity of uniform CSPs over bounded arity schemas [24]:

▶ **Theorem 28** (Theorem 1, [24]). *Assume that* FPT $\neq$ W[1]*. Then for every r.e. class $\mathbb{C}$ of structures of bounded arity the following statements are equivalent:*
1. *CSP($\mathbb{C}$, _) is in polynomial time.*
2. *p-CSP($\mathbb{C}$, _) is fixed-paramater tractable.*
3. *$\mathbb{C}$ has bounded treewidth modulo homomorphic equivalence.*

If either statement is false, then p-CSP($\mathbb{C}$, _) is W[1]*-hard.*

Direction '2 ⇒ 1' of Theorem 27 is established in [3] using arguments regarding the chase construction and applying the result from [24] on a finite portion of the chase. As concerns the lower bound (direction '1 ⇒ 2'), the proof from [3] is extremely complex: it lifts in a non-trivial way the fpt-reduction from the parameterized k-clique problem to parameterized uniform CSPs over bounded arity schemas from [24]. Lifting the reduction to the OMQ case required in particular introspection into a certain construction used in the original proof, *the Grohe database*, and modifying that construction.

Here we sketch how it is possible to establish the results from Theorem 27 using Grohe's result as a black box by employing the reduction from Theorem 20. This shows the potential of the reduction for lifting results from the CQ evaluation realm to the OMQ one. We start by establishing a counterpart of Main Result 3 for the case of bounded arities OMQs.

▶ **Theorem 29** (GDLog Bounded Arity Characterization). *Let $\mathbb{Q}$ be a r.e. class of OMQs from (GDLog, UCQ) over bounded arity schemas. Assuming that* FPT $\neq$ W[1]*:*

1. *p-OMQ($\mathbb{Q}$) is fixed-parameter tractable iff*
2. *$\mathbb{Q}_c$ has bounded tree-width iff*
3. *$\mathbb{D}_{\mathbb{Q}}$ has bounded tree-width.*

If either statement is false, then p-CSP($\mathbb{C}$, _) is W[1]*-hard.*

The strategy to prove Theorem 29 is similar to the one used to prove Theorem 19, except that this time Theorem 28 from [24] is used as a blackbox, as opposed to the unbounded arity case, where the results for uniform CSPs over unbounded arity schemas from Theorem 21 were used as a blackbox. This is the case both for showing the upper bound, direction '2 $\Rightarrow$ 1' of the theorem, and the lower bound, direction '1 $\Rightarrow$ 3' of the theorem. In the latter case, we use again our reduction from parameterized uniform CSPs to parameterized OMQs. What still needs to be shown, is the connection between the treewidth of the cover of an OMQ and the treewidth of the set of characteristic databases of the same OMQ. It follows straightaway from the proof of Lemma 22 that:

▶ **Lemma 30.** *For $Q = (\mathcal{O}, \mathbf{S}, q)$ an OMQ from (GDLog, UCQ) with schema arity at most r, and $Q_c = (\mathcal{O}, \mathbf{S}, q_c)$ its cover, it is the case that* $\mathsf{TW}(Q_c) \leq \mathsf{max}(r, \mathsf{TW}(\mathbb{D}_Q))$.

Direction '3 $\Rightarrow$ 2' of Theorem 29 follows from Lemma 30. By using Theorem 29, we can then retrieve the results from Theorem 27 similarly as we did for Theorem 24.

6 Conclusions and Future Work

In this work, we characterized the fpt border for evaluating classes of parameterized OMQs based on guarded TGDs and UCQs in the unbounded arity case. For ontologies expressed in GDLog, we provided a characterization based on new constructions, namely sets of characteristic databases and covers of an OMQ. On the way to establish this result, we introduced an fpt reduction from evaluating parameterized uniform CSPs to evaluating parameterized OMQs.

The reduction enables the lifting of results from the CSP world to the OMQ one in a modular fashion. To further showcase this, we revisited the case of OMQs over bounded arity schemas, previously addresed in [3]. For classes of such OMQs from (GDLog, UCQ) we established a new syntactic characterization of the tractability border in terms of covers and characteristic databases, while for classes of OMQs from (GTGD, UCQ), we showed how the reduction enabled a much simpler proof for the semantic characterization from [3].

Here we only considered Boolean OMQs, as the corresponding results for CQ evaluation in the unbounded arity case have also only been established in the Boolean case. As future work, we plan to extend our work to the non-Boolean case. Many results from the database world concerning efficiency of performing tasks like counting [17], enumeration [7, 14], and so on, use structural measures on the queries similar to the ones involved in the characterizations for evaluation. Thus, by generalizing our constructs to the non-Boolean case, depending on which structural measures are preserved when transitioning from sets of characteristic databases to covers of OMQs, we might be able to lift such results to the OMQ world.

References

1 Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*, 2003.

2 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.

3 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 *PODS*, pages 259–270, 2020.

4 Pablo Barceló, Cristina Feier, Carsten Lutz, and Andreas Pieris. When is ontology-mediated querying efficient? In *LICS*, pages 1–13, 2019.

5 Pablo Barceló, Diego Figueira, Georg Gottlob, and Andreas Pieris. Semantic optimization of conjunctive queries. *J. ACM*, 67(6):34:1–34:60, 2020.

6 Pablo Barceló, Andreas Pieris, and Miguel Romero. Semantic optimization in tractable classes of conjunctive queries. *SIGMOD Rec.*, 46(2):5–17, 2017.

7 Christoph Berkholz and Nicole Schweikardt. Constant delay enumeration with fpt-preprocessing for conjunctive queries of bounded submodular width. In *MFCS 2019*, volume 138, pages 58:1–58:15, 2019.

8 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.

9 Achim Blumensath. Guarded second-order logic, spanning trees, and network flows. *Logical Methods in Computer Science*, 6(1), 2010.

10 Pierre Bourhis, Marco Manna, Michael Morak, and Andreas Pieris. Guarded-based disjunctive tuple-generating dependencies. *ACM Trans. Database Syst.*, 41(4):27:1–27:45, 2016.

11 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.

12 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog±: a unified approach to ontologies and integrity constraints. In *ICDT*, pages 14–30, 2009.

13 Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

14 Nofar Carmeli and Markus Kröll. Enumeration Complexity of Conjunctive Queries with Functional Dependencies. In *(ICDT 2018)*, volume 98, pages 11:1–11:17, 2018.

15 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

16 Hubie Chen, Georg Gottlob, Matthias Lanzinger, and Reinhard Pichler. Semantic width and the fixed-parameter tractability of constraint satisfaction problems. In Christian Bessiere, editor, *IJCAI*, pages 1726–1733, 2020.

17 Hubie Chen and Stefan Mengel. A Trichotomy in the Complexity of Counting Answers to Conjunctive Queries. In *(ICDT 2015)*, volume 31, pages 110–126, 2015.

18 Bruno Courcelle. The monadic second-order logic of graphs XIV: uniformly sparse graphs and edge set quantifications. *Theor. Comput. Sci.*, 299(1-3):1–36, 2003.

19 Alin Deutsch, Alan Nash, and Jeff B. Remmel. The chase revisisted. In *PODS*, pages 149–158, 2008.

20 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM Journal on Computing*, 24:873–921, 1995.

21 Jan Foniok. Homomorphisms and structural properties of relational systems, 2007. `arXiv:0710.4477`.

22 Erich Grädel. Decision procedures for guarded logics. In *CADE*, volume 1632, 1999.

23 Erich Grädel, Colin Hirsch, and Martin Otto. Back and forth between guarded and modal logics. *ACM Trans. Comput. Log.*, 3(3):418–463, 2002.

24 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007.

25 Stijn Heymans, Jos de Bruijn, Livia Predoiu, Cristina Feier, and Davy Van Nieuwenborgh. Guarded hybrid knowledge bases. *Theory Pract. Log. Program.*, 8(3):411–429, 2008.

26 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

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.

28 Vladimir Lifschitz. Action languages, answer sets, and planning. In *APT*, pages 357–374, 1999.

29 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

30 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC*, pages 735–744, 2010.

31 Riccardo Rosati. On the decidability and complexity of integrating ontologies and rules. *J. Web Semant.*, 3(1):41–60, 2005.

32 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

Tuple-Generating Dependencies Capture Complex Values

Maximilian Marx
Knowledge-Based Systems Group, TU Dresden, Germany

Markus Krötzsch
Knowledge-Based Systems Group, TU Dresden, Germany

Abstract

We formalise a variant of Datalog that allows complex values constructed by nesting elements of the input database in sets and tuples. We study its complexity and give a translation into sets of tuple-generating dependencies (TGDs) for which the standard chase terminates on any input database. We identify a fragment for which reasoning is tractable. As membership is undecidable for this fragment, we develop decidable sufficient conditions.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Constraint and logic programming; Theory of computation → Logic and databases

Keywords and phrases terminating standard chase, existential rules, Datalog, complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.13

Funding This work is partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project number 389792660 (TRR 248, Center for Perspicuous Systems), by the Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research) in the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), and by the Center for Advancing Electronics Dresden (cfaed).

Acknowledgements We would like to thank the anonymous reviewers for their detailed feedback that led to this revised version of the paper.

1 Introduction

Complex values (also called complex objects) are formed by combining domain elements into sets, tuples, and sets and tuples of other complex values. Numerous extensions of the relational data model with complex values have been studied since the 1980s: LDL1 [37, 6] and LPS [27] support non–nested sets, whereas COL [1] can express all complex values. Several fixed point logics for complex values, as well as their tractable fragments, have also been studied [33, 36, 22]. Extensions of Datalog include Relationlog [28], Set-extended Datalog [39], IQL [3], O-Logic [25] and Higher-order Datalog [14]. Abiteboul et al. [2] provide an extensive treatment of the complex values algebra and calculus, and briefly introduce a variant of stratified Datalog with complex values, though without any formal definition.

These expressive data models have recently regained much interest. A popular example are JSON objects, which can be viewed as sets of attribute-value pairs. Such "values" are at the heart of NoSQL systems, such as CouchDB and RethinkDB, and also supported by classical RDBMS, such as PostgreSQL and MariaDB. Query languages for JSON include J-Logic [23] and RNJL [9]. Another important example are rich graph models, such as Property Graph [34, 35] and the Wikidata knowledge graph [38], which are widely used in applications. There, one often deals with sets of "annotations" (e.g., attribute–value pairs) attached to edges, which can be naturally represented as complex values. Query languages such as MARPL [30], eMARPL [29], and G-CORE [5] have been proposed for this scenario.

Graphs also bring up the need for recursive queries, even for basic tasks such as reachability. Languages like G-CORE go beyond this by supporting "paths as first class citizens" [5] in queries that return paths. Similar functionality can be realised using complex values:

▶ **Example 1.** Consider a database encoding a directed graph using facts $\mathsf{edge}(s, t)$ to denote edges from vertex s to vertex t. In our proposed formalism $\mathsf{Datalog}^{\mathrm{CV}}$, we can query for paths (represented as sets of edges, i.e., as sets of pairs of vertices) from x to y as follows:

$$\mathsf{edge}(x, y) \to \mathsf{path}(x, y, \{\!\{\langle x, y\rangle\}\!\}) \tag{1}$$

$$\mathsf{path}(x, y, P) \wedge \mathsf{edge}(y, z) \to \mathsf{path}(x, z, P \cup \{\!\{\langle y, z\rangle\}\!\}) \tag{2}$$

Intuitively, rule (1) states that whenever there is an edge from x to y, there is also a path going along that edge. Rule (2) extends a path from x to y along an edge from y to z.

Consider a database with the following facts:

$$\mathsf{edge}(a, b) \qquad \mathsf{edge}(b, c) \qquad \mathsf{edge}(a, c)$$
$$\mathsf{edge}(a, d) \qquad \mathsf{edge}(d, c) \qquad \mathsf{edge}(d, e)$$

Then we can derive the following three paths from a to c:

$$\mathsf{path}(a, c, \{\!\{\langle a, c\rangle\}\!\}) \qquad \mathsf{path}(a, c, \{\!\{\langle a, b\rangle, \langle b, c\rangle\}\!\}) \qquad \mathsf{path}(a, c, \{\!\{\langle a, d\rangle, \langle d, c\rangle\}\!\})$$

Other examples of even more complex recursive queries over complex values are found in symbolic AI, where Datalog with set values has been successfully used to reason with description logics and guarded Horn logics [32, 4, 13, 12].

Many of the above extensions involve declarative rules, which are of interest for applications such as data exchange or ontology-based query answering. Surprisingly, however, there is almost no practical support for such rules, even for limited uses of complex values. One of the most advanced systems available is *DLV-complex* [10], a logic programming engine that is not intended for database use (and only available as a 32bit-binary, i.e., limited to 4GB of memory). Hence, to realise set-based reasoning in practice, Carral et al. [13] have translated their "datalog plus sets" programs into sets of *tuple-generating dependencies* (TGDs), for which modern engines exist. Similar to Datalog, a bottom-up algorithm (the *standard chase*) can be used for reasoning, and Carral et al. show that, under certain restrictions on the order of rule applications, this approach will successfully terminate on any input database.

In this paper, we set out to generalise existing ad-hoc extensions into $\mathsf{Datalog}^{\mathrm{CV}}$, an extension of Datalog with full support for complex values. The language corresponds to the positive variant of Datalog with complex values as informally described by Abiteboul et al. [2], but we add a detailed analysis of complexity and expressivity, with a modern focus on tractable fragments and TGD-based implementation. Concretely, our contributions are as follows:

- we formalise Datalog with complex values ($\mathsf{Datalog}^{\mathrm{CV}}$) and obtain complexity bounds, showing that our formalism is intractable even for data complexity;
- we develop a translation into TGDs that preserves entailments in a certain sense, and for which the standard chase is guaranteed to terminate for all input databases and (in contrast to earlier work [13]) all strategies;
- we identify the tractable fragment (with respect to data complexity) of *bounded-cardinality programs*, which still supports non-trivial use of complex values, and develop two sufficient criteria for recognising it; and
- we show that, unlike for Datalog and TGDs, *linear rules* do not guarantee tractability.

2 Preliminaries

We consider fixed, pairwise disjoint, and countably infinite sets $\mathbf{C}$ of *constants*, $\mathbf{P}$ of *predicate names*, $\mathbf{V}$ of *variables*, and $\mathbf{N}$ of *labelled nulls*. With each predicate name $p \in \mathbf{P}$, we associate an *arity* $\mathsf{ar}(p) \in \mathbb{N}_{\geq 0}$. An *atom* is of the form $p(t_1, \ldots, t_\ell)$, where p is an ℓ-ary predicate name and $t_1, \ldots, t_\ell \in \mathbf{C} \cup \mathbf{V} \cup \mathbf{N}$ are *terms*. We may abbreviate such a list $t_1, \ldots, t_\ell$ as $\boldsymbol{t}$, and denote by $\boldsymbol{t}|_{i \mapsto s}$ the list $t_1, \ldots, t_{i-1}, s, t_{i+1}, \ldots, t_\ell$ obtained from $\boldsymbol{t}$ by replacing the i-th element by s. An atom is *null-free* if it does not contain any labelled nulls, *variable-free* if does not contain any variables, and *ground* if it is both null-free and variable-free. A *tuple-generating dependency* (TGD) or *rule* ρ is a formula of first-order logic of the form

$$\forall \boldsymbol{x}, \boldsymbol{y}.\ \varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z}.\ \psi[\boldsymbol{y}, \boldsymbol{z}] \tag{3}$$

where (i) $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ are mutually disjoint lists of variables, (ii) φ and ψ are conjunctions of null-free atoms, (iii) φ contains only variables from $\boldsymbol{x} \cup \boldsymbol{y}$, and (iv) ψ contains only variables from $\boldsymbol{x} \cup \boldsymbol{z}$.

We tacitly omit the universal quantifiers when writing TGDs, and call $\mathrm{body}(\rho) = \varphi$ the *body*, $\mathrm{head}(\rho) = \psi$ the *head*, and $\mathrm{frontier}(\rho) = \boldsymbol{y}$ the *frontier* of (3). We may treat conjunctions such as φ and ψ as sets of atoms, respectively.

A *boolean conjunctive query* (BCQ) q is a first-order sentence of the form $\exists \boldsymbol{x}.\ \varphi[x]$, where φ is a conjunction of null-free atoms. A *rule set* Σ is a finite set of TGDs. A predicate name p is an *intensional database predicate* (IDB) with respect to Σ if p occurs in the head of a rule in Σ. All other predicate names are *extensional database predicates* (EDB). An *EDB schema* is a finite set $\mathbf{P}^{\mathrm{EDB}} \subsetneq \mathbf{P}$ of predicate names. A rule set Σ is *compatible* with the EDB schema if all predicate names $p \in \mathbf{P}^{\mathrm{EDB}}$ are EDB predicates with respect to Σ. A *database instance* is a set of variable-free atoms. A *database* $\mathcal{D}$ over $\mathbf{P}^{\mathrm{EDB}}$ is a finite set of ground atoms using only predicate names from $\mathbf{P}^{\mathrm{EDB}}$. If $\mathcal{D}$ is used with rule set Σ, we always assume that $\mathcal{D}$ is over an EDB schema that Σ is compatible with.

Let $\mathcal{I}$ be a database instance, Σ a rule set, and $\mathcal{D}$ a database for Σ. Given a set A of atoms, a *homomorphism* is a function $h : A \rightarrow \mathcal{I}$ that maps terms in A to (variable-free) terms in $\mathcal{I}$, such that (i) $h(c) = c$ for all $c \in \mathbf{C}$ and (ii) $p(h(t_1), \ldots, h(t_\ell)) \in \mathcal{I}$ for all $p(t_1, \ldots, t_\ell) \in A$. A *match* of a rule ρ in $\mathcal{I}$ is a homomorphism $h : \mathrm{body}(\rho) \rightarrow \mathcal{I}$; it is *satisfied* in $\mathcal{I}$ if there is a homomorphism $h' : \mathrm{head}(\rho) \rightarrow \mathcal{I}$ with $h(x) = h'(x)$ for all $x \in \mathrm{frontier}(\rho)$. $\mathcal{I}$ *satisfies*

- a rule ρ (written $\mathcal{I} \models \rho$) if every match of ρ is satisfied,
- the rule set Σ (written $\mathcal{I} \models \Sigma$) if $\mathcal{I} \models \rho$ for all $\rho \in \Sigma$, and
- a BCQ $q = \exists \boldsymbol{x}.\ \varphi[\boldsymbol{x}]$ (written $\mathcal{I} \models q$) if there is a homomorphism $h : \varphi[\boldsymbol{x}] \rightarrow \mathcal{I}$.

$\mathcal{I}$ is a *model* of $\mathcal{D}$ and Σ (written $\mathcal{I} \models \mathcal{D}, \Sigma$) if $\mathcal{D} \subseteq \mathcal{I}$ and $\mathcal{I} \models \Sigma$. $\mathcal{D}$ and Σ *entail* a BCQ q (written $\Sigma, \mathcal{D} \models q$) if $\mathcal{I} \models q$ for all models $\mathcal{I}$ of $\mathcal{D}$ and Σ. A model $\mathcal{I} \models \mathcal{D}, \Sigma$ is *universal* if it admits homomorphisms into every model of Σ and $\mathcal{D}$. Universal models capture BCQ entailment: for q a BCQ and $\mathcal{I} \models \mathcal{D}, \Sigma$ a universal model, $\mathcal{I} \models q$ iff $\Sigma, \mathcal{D} \models q$.

Universal models can be computed by, e.g., the *standard chase* (or *restricted chase*) [7]. A *chase sequence* for a rule set Σ and database $\mathcal{D}$ is a sequence of database instances $\mathcal{D}_0, \mathcal{D}_1, \ldots$, such that: (i) $\mathcal{D}_0 = \mathcal{D}$; (ii) for every $i \geq 0$, there is a rule $\varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z}.\ \psi[\boldsymbol{y}, \boldsymbol{z}]$ in Σ with an unsatisfied match h in $\mathcal{D}_i$, and $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \psi[h'(\boldsymbol{y}), h'(\boldsymbol{z})]$, where h and h' agree on $\boldsymbol{y}$, and $h'(z) \in \mathbf{N}$ are distinct labelled nulls not occurring in $\mathcal{D}_i$ for all $z \in \boldsymbol{z}$; and (iii) if some rule ρ has a match h in some $\mathcal{D}_i$, then there is a $j > i$ such that h is satisfied in $\mathcal{D}_j$. The *chase* of $\mathcal{D}$ and Σ is the database instance $\bigcup_{i \geq 0} \mathcal{D}_i$. In general, termination of the chase can depend on the order of rule applications in step (ii) [17]; see [26] for a discussion.

3 Datalog with Complex Values

We now introduce $\mathsf{Datalog}^{\mathrm{CV}}$, an extension of Datalog in which values are terms built from tuples and sets of constants. We use sorts to represent different types of terms.

Syntax

The set $\mathcal{S}$ of *sorts* is defined inductively to contain (i) the *domain sort* Δ, (ii) for all $\tau \in \mathcal{S}$, the *set sort* $\{\tau\}$, and (iii) for all $\ell \geq 2$ and $\tau_1, \ldots, \tau_\ell \in \mathcal{S}$, the *tuple sort* $\langle \tau_1, \tau_2, \ldots, \tau_\ell \rangle$. A *subsort* of sort τ is any sort that occurs syntactically in τ, including τ itself. Every variable $v \in \mathbf{V}$ has a sort $\mathsf{sort}(v) \in \mathcal{S}$ such that the sets $\mathbf{V}_\tau = \{v \in \mathbf{V} \mid \mathsf{sort}(v) = \tau\}$ are countably infinite. The sets $\mathbf{T}_\tau$ of *terms of sort* τ are defined as follows:

1. $\mathbf{T}_\Delta \coloneqq \mathbf{C} \cup \mathbf{V}_\Delta$ (terms of the domain sort are constants or variables);
2. $\mathbf{T}_{\langle \tau_1, \ldots, \tau_\ell \rangle} \coloneqq \{\langle s_1, \ldots, s_\ell \rangle \mid s_i \in \mathbf{T}_{\tau_i} \text{ for } 1 \leq i \leq \ell\} \cup \mathbf{V}_{\langle \tau_1, \ldots, \tau_\ell \rangle}$ (terms of tuple sorts are tuples over the individual component sorts, or variables); and
3. $\mathbf{T}_{\{\tau\}} \coloneqq \{\{t_1, \ldots, t_n\} \mid n \geq 0, t_1, \ldots, t_n \in \mathbf{T}_\tau\} \cup \{(t_1 \cap t_2), (t_1 \cup t_2) \mid t_1, t_2 \in \mathbf{T}_{\{\tau\}}\} \cup \mathbf{V}_{\{\tau\}}$ (terms of set sorts are set literals over the component sort, intersections or unions of set terms of the same sort, or variables).

We also write $\{\}$ as $\emptyset$. Note that symbols like $\cup$ are overloaded to denote different functions for each sort. To avoid confusion, we sometimes use subscripts to emphasise the sort, as in $\emptyset_{\{\tau\}}$. A term is *basic* if it does not contain $\cap$ or $\cup$, and *ground* if it is variable-free.

Every predicate $p \in \mathbf{P}$ is associated with a sort $\mathsf{sort}(p)$. A *schema* $\mathbf{S} = \langle \mathbf{P}^{\mathrm{EDB}}, \mathbf{P}^{\mathrm{IDB}} \rangle$ is a partition of $\mathbf{P}$ into *EDB predicates* $\mathbf{P}^{\mathrm{EDB}}$and *IDB predicates* $\mathbf{P}^{\mathrm{IDB}}$. An *atom* for predicate $p \in \mathbf{P}$ is an expression $p(t)$ where $t \in \mathbf{T}_{\mathsf{sort}(p)}$. If p is of a tuple sort, we write $p(t_1, \ldots, t_\ell)$ instead of $p(\langle t_1, \ldots, t_\ell \rangle)$. Similarly, we may omit the outermost parentheses in set terms $(t_1 \cup t_2)$ and $(t_1 \cap t_2)$, writing $t_1 \cup t_2$ and $t_1 \cap t_2$, respectively. A $\mathsf{Datalog}^{\mathrm{CV}}$ *fact* is an atom $p(t)$, where t contains only basic ground terms. A $\mathsf{Datalog}^{\mathrm{CV}}$ *rule* is a formula of the form $\forall \boldsymbol{x}.\varphi \to \psi$ where the *body* φ and the *head* ψ are conjunctions of atoms, and $\boldsymbol{x}$ is a list of all variables in the rule. We allow φ to be empty, but require ψ to contain at least one atom. Universal quantifiers are usually omitted. A $\mathsf{Datalog}^{\mathrm{CV}}$ *program* $\mathbb{P}$ for schema $\mathbf{S}$ is a finite set of $\mathsf{Datalog}^{\mathrm{CV}}$ rules, where EDB predicates do not occur in rule heads. A $\mathsf{Datalog}^{\mathrm{CV}}$ *database* $\mathbb{D}$ for schema $\mathbf{S}$ is a finite set of facts using only EDB predicates. A $\mathsf{Datalog}^{\mathrm{CV}}$ *BCQ* is a sentence of the form $\exists \boldsymbol{x}.\varphi[\boldsymbol{x}]$, where φ is a conjunction of atoms containing only basic terms.

Already before defining the semantics of $\mathsf{Datalog}^{\mathrm{CV}}$ formally, we can foster our intuitive understanding of these definitions by considering some examples. This will also yield some useful insights into the expressive power of this formalism.

▶ **Example 2.** An operation that is commonly studied in query formalisms with complex values is the powerset function. We can capture this in atoms of the form $\mathsf{PS}_\sigma(S, P)$ expressing that a set S (of set sort $\sigma = \{\tau\}$) has the powerset P (of sort $\{\sigma\} = \{\{\tau\}\}$). We use auxiliary predicates PSU_τ of sort $\langle \tau, \{\{\tau\}\}, \{\{\tau\}\} \rangle$ where an atom $\mathsf{PSU}_\tau(t, P, Q)$ expresses, informally speaking, that $Q = \{S \cup \{t\} \mid S \in P\}$. As usual, we omit universal quantifiers.

$$\to \mathsf{PSU}_\tau(x, \emptyset, \emptyset) \tag{4}$$

$$\mathsf{PSU}_\tau(x, P, Q) \to \mathsf{PSU}_\tau(x, P \cup \{S\}, Q \cup \{S \cup \{x\}\}) \tag{5}$$

$$\to \mathsf{PS}_\sigma(\emptyset, \{\emptyset\}) \tag{6}$$

$$\mathsf{PS}_\sigma(S, P) \wedge \mathsf{PSU}_\tau(x, P, Q) \to \mathsf{PS}_\sigma(S \cup \{x\}, P \cup Q) \tag{7}$$

Rule (6) states that the powerset of $\emptyset$ is $\{\emptyset\}$ and rule (7) states that, given the powerset P of S, the powerset of $S \cup \{x\}$ is obtained from P by adding $Q \cup \{x\}$ for every $Q \in P$.

On a database containing only the constant c, the following facts are entailed:

$$\mathsf{PSU}_\tau(c,\emptyset,\emptyset) \qquad \mathsf{PSU}_\tau(c,\{\emptyset\},\{\{c\}\}) \qquad \mathsf{PSU}_\tau(c,\{\{\emptyset\}\},\{\emptyset_c\})$$
$$\mathsf{PS}_\tau(\emptyset,\{\emptyset\}) \qquad \mathsf{PS}_\tau(\{c\},\{\emptyset,\{c\}\})$$

Note that rules (4) and (5) are "unsafe": they use variables in the head that do not occur in the body. Our semantics restricts the scope of variables to a finite active domain so that this is never a problem. Using auxiliary predicates AD_τ and additional rules, we can transform any such rule into a safe rule: for every predicate $p \in \mathbf{P}^{\mathrm{EDB}}$ of sort τ, we add a rule $p(x) \to \mathsf{AD}_\tau(x)$. Furthermore, for tuple sorts $\tau = \langle\tau_1,\dots,\tau_\ell\rangle$, we add rules $\mathsf{AD}_\tau(x_1,\dots,x_\ell) \to \mathsf{AD}_{\tau_k}(x_k)$ for $1 \le k \le \ell$ and $\mathsf{AD}_{\tau_1}(x_1) \wedge \cdots \mathsf{AD}_{\tau_\ell}(x_\ell) \to \mathsf{AD}_\tau(x_1,\dots,x_\ell)$; for set sorts $\tau = \{\sigma\}$, we add rules $\mathsf{AD}_\sigma(x) \to \mathsf{AD}_\tau(\{x\})$, $\mathsf{AD}_\tau(\{x\}) \to \mathsf{AD}_\sigma(x)$, and $\mathsf{AD}_\tau(x) \wedge \mathsf{AD}_\tau(y) \to \mathsf{AD}_\tau(x \cap y) \wedge \mathsf{AD}_\tau(x \cup y)$. An unsafe variable v of sort τ in rule ρ can then be eliminated by adding an atom $\mathsf{AD}_\tau(v)$ to the body of ρ. For (4) and (5), we would thus obtain the following rules:

$$\mathsf{AD}_\tau(x) \to \mathsf{PSU}_\tau(x,\emptyset,\emptyset) \tag{8}$$
$$\mathsf{AD}_\tau(x) \wedge \mathsf{AD}_{\{\tau\}}(S) \wedge \mathsf{PSU}_\tau(x,P,Q) \to \mathsf{PSU}_\tau(x,P \cup \{S\},Q \cup \{S \cup \{x\}\}) \tag{9}$$

▶ **Example 3.** We can also define further set-related predicates and functions. The following rules show how to define predicates $\subseteq_\sigma$, $\in_\sigma$, $\notin_\sigma$, $\neq_\tau$, and $\subset_\sigma$ for a set sort σ and arbitrary sort τ, where we write infix $t_1 \diamond t_2$ instead of $\diamond(t_1,t_2)$ for better readability:

$$\to S \subseteq_\sigma S \cup T \tag{10}$$
$$S \cup \{x\} \subseteq_\sigma S \to x \in_\sigma S \tag{11}$$
$$S \cap \{x\} \subseteq_\sigma \emptyset \to x \notin_\sigma S \tag{12}$$
$$\{x\} \cap \{y\} \subseteq_{\{\tau\}} \emptyset \to x \neq_\tau y \tag{13}$$
$$S \subseteq_\sigma T \wedge x \in_\sigma T \wedge x \notin_\sigma S \to S \subset_\sigma T \tag{14}$$

We may therefore assume without loss of generality that the shortcuts defined in Examples 2 and 3 are always available in $\mathsf{Datalog}^{\mathrm{CV}}$ programs. Similarly, unless stated to the contrary, we assume throughout the paper that databases contain only facts of sort Δ or of some tuple sort $\langle\Delta,\dots,\Delta\rangle$, since all other facts can be constructed using appropriate rules.

Semantics

As usual for Datalog, we consider a *Herbrand interpretation* whose domain is the set of constants that syntactically occur in a given program and database, where we interpret constants as themselves (*unique name assumption*). To this end, we interpret sorts and ground terms with a function $\mathsf{eval}(\cdot)$. For sorts, let $\mathsf{eval}(\Delta) := \mathbf{C}$, $\mathsf{eval}(\langle\tau_1,\dots,\tau_\ell\rangle) := \mathsf{eval}(\tau_1) \times \cdots \times \mathsf{eval}(\tau_\ell)$, and $\mathsf{eval}(\{\tau\}) := \mathcal{P}(\mathsf{eval}(\tau))$ where $\mathcal{P}(S)$ denotes the set of all subsets of S. For ground terms t, we recursively define $\mathsf{eval}(t)$ as follows:

- $\mathsf{eval}(c) := c$ for $c \in \mathbf{C}$;
- $\mathsf{eval}(\{t_1,\dots,t_n\}) := \{\mathsf{eval}(t_1),\dots,\mathsf{eval}(t_n)\}$;
- $\mathsf{eval}(\langle s_1,\dots,s_\ell\rangle) := \langle\mathsf{eval}(s_1),\dots,\mathsf{eval}(s_\ell)\rangle$;
- $\mathsf{eval}(t_1 \cap t_2) := \mathsf{eval}(t_1) \cap \mathsf{eval}(t_2)$; and
- $\mathsf{eval}(t_1 \cup t_2) := \mathsf{eval}(t_1) \cup \mathsf{eval}(t_2)$.

The *domain* $\mathsf{dom}(\mathbb{P},\mathbb{D})$ of a $\mathsf{Datalog}^{\mathrm{CV}}$ program $\mathbb{P}$ and database $\mathbb{D}$ is the set of all constants that occur in $\mathbb{P}$ or $\mathbb{D}$. An *interpretation* $\mathcal{I}$ for $\mathbb{P}$ and $\mathbb{D}$ maps every predicate p to a set $p^{\mathcal{I}} \subseteq \mathsf{eval}(\mathsf{sort}(p))$ that contains only constants from $\mathsf{dom}(\mathbb{P},\mathbb{D})$. $\mathcal{I}$ *satisfies* a ground atom

$p(t)$, written $\mathcal{I} \models p(t)$, if $\mathsf{eval}(t) \in p^{\mathcal{I}}$; it *satisfies* a conjunction φ of such atoms, written $\mathcal{I} \models \varphi$, if $\mathcal{I} \models \alpha$ for all α in φ; and it *satisfies* a variable-free rule $\varphi \to \psi$, written $\mathcal{I} \models \varphi \to \psi$, if $\mathcal{I} \not\models \varphi$ or $\mathcal{I} \models \psi$. $\mathcal{I}$ *satisfies* $\mathbb{P}$ if $\mathcal{I} \models \rho'$ holds for every *ground instance* ρ' of ρ, i.e., for every variable-free rule ρ' obtained from a rule $\rho \in \mathbb{P}$ by replacing each variable $v \in \mathbf{V}_\tau$ in ρ with a ground term $t_v \in \mathsf{eval}(\tau)$ that uses only constants from $\mathsf{dom}(\mathbb{P}, \mathbb{D})$. $\mathcal{I}$ *satisfies* $\mathbb{D}$ if $\mathcal{I} \models \alpha$ for all $\alpha \in \mathbb{D}$. If $\mathcal{I}$ satisfies X, we also say that $\mathcal{I}$ is a *model* of X.

This model theory gives rise to entailment as usual: a ground fact α is entailed by $\mathbb{P}$ and $\mathbb{D}$, written $\mathbb{P}, \mathbb{D} \models \alpha$, if $\mathcal{I} \models \alpha$ for all models $\mathcal{I}$ of $\mathbb{P}$ and $\mathbb{D}$. BCQ entailment can be reduced to ground fact entailment by using BCQs as bodies of a rule with some ground head atom that is not derived by any other rule. As in the case of Datalog, entailment can be decided by considering a single *least model*, which could also be computed by a chase-like process, as the following theorem shows.

▶ **Theorem 4.** *Let* $\mathbf{S}$ *be a schema,* $\mathbb{P}$ *be a* $\mathsf{Datalog}^{CV}$ *program for* $\mathbf{S}$*, and* $\mathbb{D}$ *be a* $\mathsf{Datalog}^{CV}$ *database for* $\mathbf{S}$*. Then* $\mathbb{P}, \mathbb{D}$ *has a unique* least model $\mathcal{I}$ *such that for all ground facts,* $\mathcal{I} \models \alpha$ *iff* $\mathbb{P}, \mathbb{D} \models \alpha$*. Furthermore,* $\mathcal{I}$ *is the least fixed point of the* immediate consequence operator $T_{\mathbb{P}}$ *applied to* $\mathbb{D}$*: for a set* S *of ground facts,* $T_{\mathbb{P}}(S) \coloneqq S \cup \{\mathrm{head}(\rho') \mid \mathrm{body}(\rho') \in S$ *for a ground instance* ρ' *of* $\rho \in \mathbb{P}\}$ *is the union of* S *and the set of ground facts obtained by adding all heads of ground instances of rules that have their bodies contained in* S*.*

Complexity

The complexity of deciding entailment for a $\mathsf{Datalog}^{\mathrm{CV}}$ program $\mathbb{P}$ depends crucially on the set sorts occurring in $\mathbb{P}$, motivating the following definition.

▶ **Definition 5.** *The* set height $\mathsf{s\text{-}height}(\tau)$ *of a sort* τ *is defined recursively:*

- $\mathsf{s\text{-}height}(\Delta) \coloneqq 0$*;*
- $\mathsf{s\text{-}height}(\{\sigma\}) \coloneqq \mathsf{s\text{-}height}(\sigma) + 1$*; and*
- $\mathsf{s\text{-}height}(\langle \sigma_1, \ldots, \sigma_\ell \rangle) \coloneqq \max_{1 \le i \le \ell} \mathsf{s\text{-}height}(\sigma_i)$.

Analogously, the tuple height $\mathsf{t\text{-}height}(\tau)$ *of a sort* τ *is*

- $\mathsf{t\text{-}height}(\Delta) \coloneqq 0$*;*
- $\mathsf{t\text{-}height}(\{\sigma\}) \coloneqq \mathsf{t\text{-}height}(\sigma)$*; and*
- $\mathsf{t\text{-}height}(\langle \tau_1, \ldots, \tau_\ell \rangle) \coloneqq 1 + \max_{1 \le i \le \ell} \mathsf{t\text{-}height}(\tau_i)$.

The set height (tuple height) of a term, predicate, or variable is the set height (tuple height) of its sort, the set height (tuple height) of a schema $\mathbf{S}$ *is the largest set height (tuple height) of any predicate in* $\mathbf{S}$*, and the* height *of* $\mathbf{S}$ *is the sum of its set height and its tuple height.*

We get the following lower bounds for reasoning in $\mathsf{Datalog}^{\mathrm{CV}}$, where 0ExpTime denotes PTime. Matching upper bounds are shown in Section 4.

▶ **Theorem 6.** *Consider a* $\mathsf{Datalog}^{CV}$ *program* $\mathbb{P}$ *and a database* $\mathbb{D}$ *for schema* $\mathbf{S}$*, and a ground fact* α*. Deciding* $\mathbb{P}, \mathbb{D} \models \alpha$ *is* kExpTime*-hard with respect to the size of* $\mathbb{D}$ *(*data complexity*) and* $(k+2)$ExpTime*-hard with respect to the size of* $\mathbb{D}$ *and* $\mathbb{P}$ *(*combined complexity*), where* $k = \mathsf{s\text{-}height}(\mathbf{S})$.

If $\mathsf{t\text{-}height}(\mathbf{S})$ *is bounded, combined complexity is only* $(k+1)$ExpTime*-hard.*

Since Datalog has bounded tuple height 1 and set height 0, our complexity bounds (PTime data and ExpTime combined) are optimal in this case [16].

Proof. For data complexity, we reduce from the halting problem for k-exponentially time-bounded (in the size of $\mathbb{D}$) Turing machines on the empty tape, where $\mathbb{P}$ encodes the Turing machine. The main challenge is the construction of the tape, as Turing machines can be simulated already in Datalog [16].

Assuming that a linear order $s_1 < s_2 < \cdots < s_\ell$ for sort τ is encoded using unary predicates $\mathsf{first}_\tau(s_1)$ and $\mathsf{last}_\tau(s_\ell)$ and a binary predicate $\mathsf{next}_\tau(s_i, s_{i+1})$, the rules in Figure 1 derive an exponentially longer chain for sort $\sigma = \{\tau\}$. Let $\mathcal{S} = \{s_1, s_2, \ldots, s_\ell\}$.

Intuitively, the program enumerates the power set $\mathcal{P}(\mathcal{S})$ of $\mathcal{S}$ in lexicographic order, producing the exponentially longer linear order $\emptyset < \{s_1\} < \{s_1, s_2\} < \cdots < \{s_1, s_2, \ldots, s_\ell\} < \{s_2\} < \cdots < \{s_2, \ldots, s_\ell\} < \cdots < \{s_\ell\}$. A fact $\mathsf{mkStep}_\sigma(S, t, x)$ will lead to next_σ-facts corresponding to the segment $S \cup \{x\} < \cdots < S \cup \{s_\ell\}$, with t tracking the least element not contained in S. Similarly, a fact $\mathsf{step}_\sigma(S, t, x, T)$ corresponds to the segment $S < S \cup \{x\} < \cdots < T$, where t is again the least element not contained in S.

To see that rules (15)–(19) are correct, we show the following claim: for any set $S \subseteq \mathcal{S}$ and any $t, x \in \{s \in \mathcal{S} \mid \forall y \in S.\ y < t \wedge y < x\}$, the fact $\mathsf{mkStep}_\sigma(S, t, x)$ will result in facts $\mathsf{next}_\sigma(S \cup \{x\}, \ldots), \ldots, \mathsf{next}_\sigma(\ldots, S \cup \{s_\ell\})$ corresponding to the segment $S \cup \{x\} < \cdots < S \cup \{s_\ell\}$ of the lexicographic order on $\mathcal{P}(\mathcal{S})$, and furthermore a fact $\mathsf{step}_\sigma(S, t, x, S \cup \{s_\ell\})$.

We show the claim by induction on the third position in mkStep_σ-facts. For the base case, consider $\mathsf{mkStep}_\sigma(S, s_\ell, s_\ell)$. From rule (16), we immediately get $\mathsf{next}_\sigma(S, S \cup \{s_\ell\})$, and rule (17) derives $\mathsf{step}_\sigma(S, s_\ell, s_\ell, S \cup \{s_\ell\})$. For the induction step, suppose that x has the direct successor y. Rule (18) derives the facts $\mathsf{mkStep}_\sigma(S \cup \cup\, x, y, y)$ and $\mathsf{mkStep}_\sigma(S, t, y)$. By the induction hypothesis, we therefore derive next_σ and step_σ-facts corresponding to the segments $S \cup \{x\} < S \cup \{x, y\} < \cdots < S \cup \{x, s_\ell\}$ of successors containing x, and $S \cup \{y\} < \cdots < S \cup \{s_\ell\}$ of successors not containing x. Since y is the direct successor of x, the immediate successor of $S \cup \{x, s_\ell\}$ is $S \cup \{y\}$, and rule (19) therefore connects the two segments. Lastly, whenever $t = x$, rule (16) makes $S \cup \{x\}$ the direct successor of S. Such mkStep_σ-facts are derived by rule (15), where we immediately see that this is correct, since $\{s_1\}$ is indeed the direct successor of $\emptyset_\sigma$, and by rule (18), where we consider $S \cup \{x\}$. Since y is the direct successor of x, this makes $S \cup \{x, y\}$ the direct successor of $S \cup \{x\}$.

Since no other mkStep_σ-facts are derived, no cycles in next_σ-facts can appear, and thus we indeed obtain a linear order. Correctness then follows since rule (15) derives $\mathsf{first}_\sigma(\emptyset_\sigma)$, $\mathsf{mkStep}_\sigma(\emptyset_\sigma, s_1, s_1)$, and $\mathsf{last}_\sigma(\{s_\ell\})$.

By instantiating these rules for multiple set sorts, we can thus construct k-exponentially long linear orders from a linear order in the input database, which we can use as a Turing tape for our simulation [16].

For combined complexity, we use a tuple sort of nested tuples of height t and constants at the lowest level. We then construct a chain of doubly-exponential length in t by ordering tuples at each of the t levels lexicographically, starting from the length-ℓ chain of database constants. If tuples have arity $\geq a$, this yields a chain of length $\geq \ell^{a^t}$. The construction then proceeds as before, using this chain as a basis for an even longer chain of sets. Note that if t is bounded by a constant (but a is not), then ℓ^{a^t} is merely single exponential. ◀

4 From Complex Values to TGDs

In this section, we reduce query answering over $\mathsf{Datalog}^{\mathrm{CV}}$ programs and $\mathsf{Datalog}^{\mathrm{CV}}$ databases to query answering over sets of TGDs and (unsorted) databases for which the standard chase terminates reliably (i.e., under all strategies).

We build upon prior work by Carral et al. [13], which established a translation from "Datalog with Sets" Datalog(S) to sets of TGDs. Datalog(S) can be seen as the restriction of $\mathsf{Datalog}^{\mathrm{CV}}$ to schemas where every predicate has sort Δ, $\{\Delta\}$, or $\langle \tau_1, \ldots, \tau_\ell \rangle$ with $\tau_i \in \{\Delta, \{\Delta\}\}$ for $1 \leq \ell$, i.e., where every position is either an element of the domain or a set over such elements. The key idea is to represent sets by recursively constructing them

$$\mathsf{first}_\tau(x) \wedge \mathsf{last}_\tau(z) \to \mathsf{first}_\sigma(\emptyset_\sigma) \wedge \mathsf{mkStep}_\sigma(\emptyset_\sigma, x, x) \wedge \mathsf{last}_\sigma(\{z\}) \quad (15)$$

$$\mathsf{mkStep}_\sigma(S, t, t) \to \mathsf{next}_\sigma(S, S \cup \{t\}) \quad (16)$$

$$\mathsf{mkStep}_\sigma(S, t, x) \wedge \mathsf{last}_\tau(x) \to \mathsf{step}_\sigma(S, t, x, S \cup \{x\}) \quad (17)$$

$$\mathsf{mkStep}_\sigma(S, t, x) \wedge \mathsf{next}_\tau(x, y) \to \mathsf{mkStep}_\sigma(S \cup \{x\}, y, y) \wedge \mathsf{mkStep}_\sigma(S, t, y) \quad (18)$$

$$\mathsf{mkStep}_\sigma(S, t, x) \wedge \mathsf{next}_\tau(x, y) \wedge \mathsf{step}_\sigma(S \cup \{x\}, y, y, X) \wedge \mathsf{step}_\sigma(S, t, y, Z) \to \mathsf{next}_\sigma(X, S \cup \{y\}) \wedge \mathsf{step}_\sigma(S, t, x, Z) \quad (19)$$

Figure 1 A $\mathsf{Datalog}^{\mathrm{CV}}$ program for a set sort $\sigma = \{\tau\}$ deriving an exponentially long linear order.

$$\textstyle\bigwedge_{i=1}^{\ell} \mathsf{sort}_{\tau_i}(x_i) \to \exists z.\mathsf{tuple}_\pi(z, x_1, \ldots, x_\ell) \wedge \mathsf{sort}_\pi(z) \quad (20)$$

$$\to \exists V.\mathsf{empty}_\sigma(V) \wedge \mathsf{sort}_\sigma(V) \wedge \mathsf{done}_\sigma(V) \quad (21)$$

$$\mathsf{done}_\sigma(V) \wedge \mathsf{sort}_\tau(x) \to \exists W.\mathsf{SU}_\sigma(x, V, W) \wedge \mathsf{sort}_\sigma(W) \wedge \mathsf{todo}_\sigma(W, W) \quad (22)$$

$$\mathsf{todo}_\sigma(V, W) \wedge \mathsf{SU}_\sigma(x, U, V) \to \mathsf{SU}_\sigma(x, W, W) \wedge \mathsf{todo}_\sigma(U, W) \quad (23)$$

$$\mathsf{todo}_\sigma(V, W) \wedge \mathsf{empty}_\sigma(V) \to \mathsf{done}_\sigma(W) \quad (24)$$

Figure 2 TGDs for axiomatising $\mathsf{Datalog}^{\mathrm{CV}}$ sorts and terms; we instantiate (20) for sorts $\pi = \langle \tau_1, \ldots, \tau_\ell \rangle$, and (21)–(24) for sorts $\sigma = \{\tau\}$.

as unions of singletons and smaller sets. While our translation shares this approach, an important difference is that our translation produces sets of TGDs for which the standard chase terminates on any database, irregardless of the order of rule applications, while the translation of Carral et al. relies on the prioritisation of certain rules to ensure termination.

The translation consists of two parts: a set of auxiliary rules that axiomatise the semantics of set functions and predicates for all sorts used in the program, and, for every given rule, a rewritten rule that uses these defined predicates instead of set functions. After the translation, all $\mathsf{Datalog}^{\mathrm{CV}}$ terms are represented by individual nulls or constants. For example, facts $\mathsf{tuple}_\pi(r, t_1, \ldots, t_\ell)$ express that r represents the tuple $\langle t_1, \ldots, t_\ell \rangle$ of sort π. To encode sets, we use facts $\mathsf{SU}_\sigma(x, S, T)$ ("singleton union"), which can be read as "$\{x\} \cup_\sigma S = T$." Together with a representative $c_{\emptyset_\sigma}$ for $\emptyset_\sigma$, this allows us to represent every set $\{e_1, \ldots, e_n\}$ by a term c_n for which we give a list of facts $\mathsf{SU}_\sigma(e_1, c_{\emptyset_\sigma}, c_1), \ldots, \mathsf{SU}_\sigma(e_n, c_{n-1}, c_n)$. Note that this representation is not unique; in general we have indeed multiple representatives for every set. This may, in turn, lead to multiple representatives for the same tuple if sets occur somewhere in the component sorts. To ensure that this does not pose problems, our translated programs will include congruence rules that propagate derived facts between different representatives of the same value.

The rules for axiomatising these basic predicates are as given in Figure 2, which also defines $\mathsf{empty}_\sigma(t)$ ("t represents $\emptyset_\sigma$") and $\mathsf{sort}_\tau(t)$ ("t represents a term of sort τ"). We assume (and will ensure) that facts for sort_Δ are defined for the constants in the active domain. Rule (20) creates representatives for π-tuples. The remaining rules (21)–(24) create representatives for sets by generalising an approach that we outlined in our previous work [26] to arbitrary set sorts $\sigma = \{\tau\}$: Rule (21) creates a unique representative for $\emptyset_\sigma$, which is immediately marked as "done". Given any "done" set V and an element x of sort τ, rule (22) introduces a representative W for $S \cup \{x\}$, which is marked as "todo". Then rule (23) derives facts representing $W = U \cup \{x\}$ for all sets $U \subseteq V$ and all $x \in U$. Only when all such facts

$$\mathsf{sort}_\sigma(V) \wedge \mathsf{sort}_\sigma(W) \to \mathsf{ckSub}_\sigma(V,V,W) \tag{25}$$
$$\mathsf{ckSub}_\sigma(U,V,W) \wedge \mathsf{SU}_\sigma(x,U',U) \wedge \mathsf{SU}_\sigma(x,V,V) \to \mathsf{ckSub}_\sigma(U',V,W) \tag{26}$$
$$\mathsf{ckSub}_\sigma(U,V,W) \wedge \mathsf{empty}_\sigma(U) \to \mathsf{subset}_\sigma(V,W) \tag{27}$$
$$\mathsf{subset}_\sigma(V,W) \wedge \mathsf{subset}_\sigma(W,V) \to \mathsf{eq}_\sigma(V,W) \tag{28}$$
$$\mathsf{empty}_\sigma(V) \wedge \mathsf{sort}_\sigma(W) \to \mathsf{U}_\sigma(V,W,W) \tag{29}$$
$$\mathsf{SU}_\sigma(x,W,W') \wedge \mathsf{U}_\sigma(V,W',U) \wedge \mathsf{SU}_\sigma(x,V,V') \to \mathsf{U}_\sigma(V',W,U) \tag{30}$$
$$\mathsf{sort}_\sigma(V) \wedge \mathsf{sort}_\tau(x) \to \mathsf{ckNIn}_\sigma(V,x,V) \tag{31}$$
$$\mathsf{ckNIn}_\sigma(U,x,V) \wedge \mathsf{SU}_\sigma(y,U',U) \wedge \mathsf{NEq}_\tau(x,y) \to \mathsf{ckNIn}_\sigma(U',x,V) \tag{32}$$
$$\mathsf{ckNIn}_\sigma(U,x,V) \wedge \mathsf{empty}_\sigma(U) \to \mathsf{NIn}_\sigma(x,V) \tag{33}$$
$$\mathsf{tuple}_\pi(z,x_1,\dots,x_\ell) \wedge \mathsf{tuple}_\pi(z',y_1,\dots,y_\ell) \wedge \mathsf{NEq}_{\tau_i}(x_i,y_i) \to \mathsf{NEq}_\pi(z,z') \tag{34}$$
$$\mathsf{SU}_\sigma(x,V,V) \wedge \mathsf{NIn}_\sigma(x,W) \to \mathsf{NEq}_\sigma(V,W) \wedge \mathsf{NEq}_\sigma(W,V) \tag{35}$$
$$\mathsf{empty}_\sigma(V) \wedge \mathsf{sort}_\sigma(W) \to \mathsf{I}_\sigma(V,W,V) \tag{36}$$
$$\mathsf{I}_\sigma(U,V,W) \wedge \mathsf{SU}_\sigma(x,U,U') \wedge \mathsf{NIn}_\sigma(x,V) \to \mathsf{I}_\sigma(U',V,W) \tag{37}$$
$$\mathsf{I}_\sigma(U,V,W) \wedge \mathsf{SU}_\sigma(x,U,U') \wedge \mathsf{SU}_\sigma(x,V,V') \wedge \mathsf{SU}_\sigma(x,W,W') \to \mathsf{I}_\sigma(U',V',W') \tag{38}$$

Figure 3 TGDs for axiomatising $\mathsf{Datalog}^{\mathrm{CV}}$ sort functions and predicates; we instantiate (34) for sorts $\pi = \langle \tau_1, \dots, \tau_\ell \rangle$ and all $1 \leq i \leq \ell$, and all other rules for sorts $\sigma = \{\tau\}$.

have been derived is W marked "done" (rule (24)). But since rule (22) is only applicable for "done" sets, it will always be blocked for sets V and elements x with $x \in V$. Thus, it can only be applied finitely often, and termination is therefore guaranteed.

The essential correctness claim for these rules is as follows:

▶ **Lemma 7.** *Let Σ be the set of rules (21)–(24) for a sort $\sigma = \{\tau\}$. For every database $\mathcal{D}$ that contains only facts of the form $\mathsf{sort}_\tau(c)$, every standard chase sequence over Σ and $\mathcal{D}$ terminates after $O(2^{|\mathcal{D}|})$ many steps, producing a finite result $\mathcal{I}$ that is unique up to isomorphism. For this result $\mathcal{I}$ and the sets $S_t \coloneqq \{c \mid \mathsf{SU}_\sigma(c,t,t) \in \mathcal{I}\}$, we have:*

- *if $\mathsf{empty}_\sigma(t) \in \mathcal{I}$ then $S_t = \emptyset$,*
- *if $\mathsf{SU}_\sigma(c,t,u) \in \mathcal{I}$ then $S_u = S_t \cup \{c\}$, and*
- *$\{S_t \mid \mathsf{sort}_\sigma(t) \in \mathcal{I}\}$ is the powerset of $\{c \mid \mathsf{sort}_\tau(c) \in \mathcal{D}\}$.*

To complete the translation, we use further facts $\mathsf{eq}_\sigma(t_1,t_2)$ ("$t_1 = t_2$"), $\mathsf{U}_\sigma(t_1,t_2,t)$ ("$t_1 \cup t_2 = t$"), and $\mathsf{I}_\sigma(t_1,t_2,t)$ ("$t_1 \cap t_2 = t$"), axiomatised in Figure 3. Expressing eq is necessary since our modelling in Figure 2 may lead to several representatives for the same set. To reconcile this, we compute facts for subset_σ: for every pair $\langle V, W \rangle$ of sets (25), we iterate over elements of V (26) to verify that they are in W, until all elements have been processed (27). Equality follows from mutual inclusion (28).

Unions are defined from singleton unions by a simple recursion (29)–(30). To compute intersections, we define inequality NEq_τ (for all sorts τ) and non-containment NIn_σ (for set sorts σ) by mutual recursion (31)–(35). Facts for NEq_Δ will be defined explicitly for constants of the active domain by our translation. Rules (31)–(33) are similar to rules (25)–(27). Deriving inequalities for tuples and sets is then straightforward (34)–(35). Finally, intersections are defined starting from the empty set (36), and recursively adding elements that are missing/present in the other set (37)/(38). Reusing the notation S_t from Lemma 7, we state the essential correctness results for this translation:

▶ **Lemma 8.** *Let Σ be the set of rules (21)–(24) and (25)–(30) for a sort $\sigma = \{\!\{\tau\}\!\}$. For every database $\mathcal{D}$ that contains only facts of the form $\mathsf{sort}_\tau(c)$, the standard chase over Σ and $\mathcal{D}$ produces a finite result $\mathcal{I}$ that is unique up to isomorphism, such that:*

- $\mathsf{eq}_\sigma(t,u) \in \mathcal{I}$ *iff* $S_t = S_u$,
- $\mathsf{U}_\sigma(t,u,v) \in \mathcal{I}$ *implies* $S_t \cup S_u = S_v$, *and*
- *for all* $\mathsf{sort}_\sigma(t), \mathsf{sort}_\sigma(u) \in \mathcal{I}$, *there is some fact* $\mathsf{U}_\sigma(t,u,v) \in \mathcal{I}$.

▶ **Lemma 9.** *Let Σ be the set of rules (21)–(24), (31)–(33), and (35)–(38) for a sort $\sigma = \{\!\{\tau\}\!\}$. For every database $\mathcal{D}$ of the form $\{\mathsf{sort}_\tau(c) \mid c \in \Pi\} \cup \{\mathsf{NEq}_\tau(c,d) \mid c \neq d; c,d \in \Pi\}$ for some finite set $\Pi \subseteq \mathbf{C}$ of constants, the standard chase over Σ and $\mathcal{D}$ produces a finite result $\mathcal{I}$ that is unique up to isomorphism, such that:*

- $\mathsf{NIn}_\sigma(c,t) \in \mathcal{I}$ *iff* $c \notin S_t$,
- $\mathsf{NEq}_\sigma(t,u) \in \mathcal{I}$ *iff* $S_t \neq S_u$,
- $\mathsf{I}_\sigma(t,u,v) \in \mathcal{I}$ *implies* $S_t \cap S_u = S_v$, *and*
- *for all* $\mathsf{sort}_\sigma(t), \mathsf{sort}_\sigma(u) \in \mathcal{I}$, *there is some fact* $\mathsf{I}_\sigma(t,u,v) \in \mathcal{I}$.

To translate rules in $\mathbb{P}$, we associate with every non-constant term $t \in \mathbf{T}_\tau$ a distinct variable $x_t \in \mathbf{V}_\tau$. Let $\mathsf{tr}(t) \coloneqq t$ if $t \in \mathbf{C}$ and set $\mathsf{tr}(t) \coloneqq x_t$ otherwise, and define a set of atoms $\mathsf{flat}(t)$ for terms t of sort τ recursively as follows:

- if $t \in \mathbf{C} \cup \mathbf{V}_\tau$ then $\mathsf{flat}(t) \coloneqq \{\mathsf{sort}_\tau(\mathsf{tr}(t))\}$;
- if $t = \langle t_1, \dots, t_\ell \rangle$ then $\mathsf{flat}(t) \coloneqq \{\mathsf{tuple}_\tau(x_t, \mathsf{tr}(x_{t_1}), \dots, \mathsf{tr}(x_{t_\ell}))\} \cup \bigcup_{i=1}^{\ell} \mathsf{flat}(t_i)$;
- if $t = \emptyset_\tau$ then $\mathsf{flat}(t) \coloneqq \{\mathsf{empty}(x_t)\}$;
- if $t = \{\!\{t_1, \dots, t_n\}\!\}$ then $\mathsf{flat}(t) \coloneqq \{\mathsf{SU}(\mathsf{tr}(t_1), x_s, x_t)\} \cup \mathsf{flat}(t_1) \cup \mathsf{flat}(s)$ for $s = \{\!\{t_2, \dots, t_n\}\!\}$ (which can be empty);
- if $t = t_1 \cup t_2$ then $\mathsf{flat}(t) \coloneqq \{\mathsf{U}_\tau(\mathsf{tr}(t_1), \mathsf{tr}(t_2), x_t)\} \cup \mathsf{flat}(t_1) \cup \mathsf{flat}(t_2)$; and
- if $t = t_1 \cap t_2$ then $\mathsf{flat}(t) \coloneqq \{\mathsf{I}_\tau(\mathsf{tr}(t_1), \mathsf{tr}(t_2), x_t)\} \cup \mathsf{flat}(t_1) \cup \mathsf{flat}(t_2)$.

To translate a rule ρ to a TGD $\mathsf{tr}(\rho)$, replace each term t in ρ by $\mathsf{tr}(t)$ and add $\mathsf{flat}(t)$ to the body of ρ. For a fact $p(t)$, let $\mathsf{tr}(p(t)) \coloneqq \{p(\mathsf{tr}(t))\} \cup \mathsf{flat}(t)$, and for a conjunction $\varphi = \bigwedge_{i=1}^{n} \varphi_i$, let $\mathsf{tr}(\varphi) \coloneqq \bigcup_{i=1}^{n} \mathsf{tr}(\varphi_i)$. A BCQ $\exists \boldsymbol{x}.\varphi[\boldsymbol{x}]$ is translated into the BCQ obtained by existentially quantifying all variables in $\mathsf{tr}(\varphi)$.

For $\mathsf{Datalog}^{\mathrm{CV}}$ program $\mathbb{P}$, let $\mathsf{tr}(\mathbb{P})$ consist of (1) $\mathsf{tr}(\rho)$ for all $\rho \in \mathbb{P}$, (2) all rules in Figures 2 and 3, instantiated for all subsorts of sorts in $\mathbb{P}$, (3) the *congruence rules* $\to \mathsf{eq}(x,x)$ and $p(\boldsymbol{x}) \wedge \mathsf{eq}(x_i, x_i') \to p(\boldsymbol{x}|_{i \mapsto x_i'})$ for all predicates p and all $1 \leq i \leq \mathsf{ar}(p)$, and (4) the facts $\{\mathsf{sort}_\Delta(c) \mid c \in \Pi\} \cup \{\mathsf{NEq}_\Delta(c,d) \mid c \neq d; c,d \in \Pi\}$ for Π the set of constants in $\mathbb{P}$. Likewise, let $\mathsf{tr}(\mathbb{D})$ consist of all facts obtained by uniformly replacing, for each $\mathsf{Datalog}^{\mathrm{CV}}$ fact $\alpha \in \mathbb{D}$, the variables in $\mathsf{tr}(\alpha)$ with fresh constants. We note that tr can be computed in polynomial time (even in logarithmic space) in all cases. We obtain the following correctness result:

▶ **Theorem 10.** *For every $\mathsf{Datalog}^{CV}$ program $\mathbb{P}$ and database $\mathbb{D}$ for $\mathbb{P}$:*

1. *for every $\mathsf{Datalog}^{CV}$ BCQ q:* $\mathbb{P}, \mathbb{D} \models q$ *iff* $\mathsf{tr}(\mathbb{P}), \mathsf{tr}(\mathbb{D}) \models \mathsf{tr}(q)$,
2. *for $k = \mathsf{s\text{-}height}(\mathbb{P})$, every standard chase sequence for $\mathsf{tr}(\mathbb{P})$ over $\mathsf{tr}(\mathbb{D})$ terminates in a number of steps that is k-exponential in the size of $\mathbb{D}$ and $(k+2)$-exponential in the size of $\mathbb{P}$. Assuming bounded tuple height, it is $(k+1)$-exponential in the size of $\mathbb{P}$.*

Proof sketch. The correctness of the transformation has been discussed before. For the complexity, we estimate the maximal number of complex terms that may be required. For a set height k, we find that this is in $O(2^{2^{\cdot^{\cdot^{2^{a^t}}}}})$, where there are k occurrences of 2, a is the maximal arity of tuples, and t is the tuple height. Our simulation admits up to $m!$ many representations of each set of size m, corresponding to the order of adding elements,

which is still exponential in a polynomial over m since $m! < m^m = 2^{m \log m}$. The number of facts and rule applications in the chase for $\mathsf{tr}(\mathbb{P})$ over $\mathsf{tr}(\mathbb{D})$ is polynomial in the resulting $(k+2)$-exponential number of terms, since every fact is completely determined by a fixed number of complex terms (defined by our translation rather than by the input). ◀

Although this matches the lower complexity bounds of Theorem 6, the above translation is not efficient, already since the rules in Figure 2 create all possible terms over the given sorts. This can instead be done "on demand" if we add additional premises, e.g., $\mathsf{mkSU}(x, V)$ in rule (22). For any rule $\varphi \to \psi$ with a body atom $\mathsf{SU}(x, V, W) \in \varphi$, we can then create an additional rule $\varphi \setminus \{\mathsf{SU}(x, V, W)\} \to \mathsf{mkSU}(x, V)$ to trigger the computation of a relevant fact for $\mathsf{SU}(x, V, W)$. If several auxiliary predicates occur, they are ordered and computed successively, in the manner of the *magic sets* transformation [2].

5 A Tractable Fragment

We now turn our attention to a fragment of $\mathsf{Datalog}^{\mathrm{CV}}$ that still supports sets but keeps reasoning tractable. To this end, we require that the cardinality of any set derived by the program on any database remains bounded by a fixed k. Although it is generally undecidable whether a program has this property, we develop sufficient conditions for bounded cardinality.

▶ **Definition 11.** *A* $\mathsf{Datalog}^{CV}$ *ground fact* α *has* k-bounded cardinality *if all set terms occurring in* α *are of the form* $\{t_1, \dots, t_n\}$ *with* $n \leq k$. *A* $\mathsf{Datalog}^{CV}$ *program* $\mathbb{P}$ *has* k-bounded cardinality *if, for any database* $\mathbb{D}$*, all ground facts* α *with* $\mathbb{P}, \mathbb{D} \models \alpha$ *have* k*-bounded cardinality.* $\mathbb{P}$ *has* bounded cardinality *if it has* k*-bounded cardinality for some* $k \in \mathbb{N}$.

Note that k-bounded cardinality implies k'-bounded cardinality for all $k' \geq k$. Also note that programs containing unsafe set variables generally do not have bounded cardinality, since such variables range over all possible sets over the active domain.

▶ **Example 12.** The following program $\mathbb{P}$ has 2-bounded cardinality.

$$\mathsf{e}(x) \to \mathsf{s}(\{x\}) \tag{39}$$

$$\mathsf{s}(X) \wedge \mathsf{s}(Y) \to \mathsf{p}(X \cup Y) \tag{40}$$

We obtain a program $\mathbb{P}'$ that does not have bounded cardinality if we replace (40) by (41).

$$\mathsf{s}(X) \wedge \mathsf{s}(Y) \to \mathsf{s}(X \cup Y) \tag{41}$$

Indeed, s is now closed under unions, hence it contains a set whose cardinality is the number of $e(x)$ facts, which depends on the size of the database.

▶ **Theorem 13.** *Every* k*-bounded cardinality* $\mathsf{Datalog}^{CV}$ *program* $\mathbb{P}$ *and database* $\mathbb{D}$ *for* $\mathbb{P}$ *can be translated into a Datalog program* $\mathsf{dl}(\mathbb{P})$ *and a database* $\mathsf{dl}(\mathbb{D})$ *such that:*

1. *for every ground* $\mathsf{Datalog}^{CV}$ *fact* α*:* $\mathbb{P}, \mathbb{D} \models \alpha$ *iff* $\mathsf{dl}(\mathbb{P}), \mathsf{dl}(\mathbb{D}) \models \mathsf{dl}(\alpha)$,
2. *the translations* $\mathsf{dl}(\cdot)$ *are* PTIME*-computable with respect to* k *and the size of the input and* EXPTIME*-computable with respect to the height of the schema of* $\mathbb{P}$.

Proof. Let $\mathbb{P}$ be a k-bounded cardinality $\mathsf{Datalog}^{\mathrm{CV}}$ program. The translation eliminates complex sorts by increasing the arity of predicates: we replace every position of a set sort $\tau = \{\sigma\}$ by k positions of sort σ, filling up the remaining positions with $\sqcup_\tau$ to represent sets of cardinality less than k, and every position of some tuple sort $\pi = \langle \tau_1, \tau_2, \dots, \tau_\ell \rangle$ is

replaced by ℓ positions of sorts $\tau_1, \tau_2, \ldots, \tau_\ell$, respectively. Repeating this process leads to predicates in which every position is of the domain sort. Tuple terms $\langle t_1, t_2, \ldots, t_\ell \rangle$ are easily translated, as each individual term t_i is simply placed in one of the new positions. Basic set terms are translated similarly, but for compound set terms, the situation is a bit more complicated: Similarly to the translation into TGDs in Section 4, we add $3k$-ary atoms $\mathsf{U}_\tau(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t})$ and $\mathsf{I}_\tau(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t})$ to rules containing $t_1 \cup_\tau t_2$ and $t_1 \cap_\tau t_2$, respectively, where the individual terms are replaced by k-tuples of terms; and use the variables $\boldsymbol{t}$ in the positions corresponding to the original term. Analogously, terms $\boldsymbol{\mathcal{P}}(t)$ are replaced by $2k$-ary facts $\mathsf{PS}_\sigma(\boldsymbol{t}, \boldsymbol{p})$, where $\boldsymbol{p}$ is the k-tuple of variables corresponding to the evaluation of the term. We can then simulate the behaviour of these set operations using further rules. Since all sets have k-bounded cardinality, no more than k of the first $2k$ positions in U_τ can be distinct from $\sqcup_\tau$, and no more than $\lfloor \log_2 k \rfloor$ of the first k positions in PS_σ can be distinct from $\sqcup_\sigma$, respectively, whereas all of the first $2k$ positions of I_τ may be distinct from $\sqcup_\tau$. In either case, however, the number of rules needed to simulate these operations depends polynomially on k, and is exponential in the height of the schema of $\mathbb{P}$. Applying these transformations to $\mathbb{P}$, $\mathbb{D}$, and q, respectively, we thus obtain $\mathsf{dl}(\mathbb{P})$, $\mathsf{dl}(\mathbb{D})$, and $\mathsf{dl}(q)$, in which all positions are of the domain sort, i.e., $\mathsf{dl}(\mathbb{P})$ is a Datalog program. ◀

If we consider schemas of unbounded height, then the arity of the required Datalog predicates may grow exponentially, though it remains bounded in terms of the database size.

▶ **Corollary 14.** *Every bounded cardinality* $\mathsf{Datalog}^{CV}$ *program has* PTIME*-complete data complexity and* 2EXPTIME*-complete combined complexity.*

Proof sketch. Upper bounds follow from Theorem 13. PTIME-hardness is inherited from plain Datalog. For the 2EXPTIME-hardness, it suffices to consider a signature without sets. By ordering (nested) tuples, we can construct a doubly-exponentially long chain as in the proof of Theorem 6. Simulating a Turing machine on this chain is again standard. ◀

The next two results establish that bounded cardinality is undecidable in general but becomes decidable for a fixed bound k.

▶ **Theorem 15.** *It is undecidable whether a* $\mathsf{Datalog}^{CV}$ *program has bounded cardinality.*

Proof. Let $\mathcal{M}$ be a deterministic Turing machine. We construct a $\mathsf{Datalog}^{\mathrm{CV}}$ program $\mathbb{P}$ such that $\mathbb{P}$ has bounded cardinality iff $\mathcal{M}$ halts on the empty tape. We consider a schema consisting of a unary predicate first and a binary predicate next. We intend databases to encode initial segments of a linear order using these predicates, but we need to ensure that $\mathbb{P}$ does not produce arbitrarily large sets on invalid encodings in a database. Hence, we collect all predecessors of an element in a set: this allows us to detect cycles and untangles elements with multiple predecessors. The next two rules realise this ($\mathsf{stepped}$ is explained below):

$$\mathsf{first}(x) \to \mathsf{step}(\emptyset, \{x\}) \wedge \mathsf{lift}(x, \{x\}) \tag{42}$$

$$\mathsf{next}(x, y) \wedge \mathsf{lift}(x, X) \wedge y \notin X \wedge \mathsf{stepped}(X) \to \mathsf{step}(X, X \cup \{y\}) \wedge \mathsf{lift}(y, X \cup \{y\}) \tag{43}$$

To axiomatise $y \notin X$ without enforcing unbounded cardinality, we use rule (12), but replace rule (10) by $\to \emptyset \subseteq \emptyset$, which is the only $\subseteq$-entailment needed for (12).

In addition, $\mathbb{P}$ contains rules that simulate $\mathcal{M}$ on the empty tape, constructing a suitable grid using step-facts, starting at $\emptyset$. These rules do not require any further sets besides those used as grid points. Once a valid transition of $\mathcal{M}$ has been performed, the used set is marked as $\mathsf{stepped}$, which allows $\mathbb{P}$ to extend the linear order by one further step. If the input contains

cycles, rule (43) becomes inapplicable, as $y \notin X$ will not hold. It might happen that multiple linear orders are derived, e.g., when an element has more than one next-successors. In this case, $\mathbb{P}$ merely performs multiple simulations in parallel. If $\mathcal{M}$ halts on the empty tape after k steps, then, for every simulation, there will only be k facts of the form $\mathsf{stepped}(X)$. Each of these sets X contains at most $k-1$ elements, so $\mathbb{P}$ has bounded cardinality. If, however, $\mathcal{M}$ does not halt on the empty tape, the databases $\mathbb{D}_i = \{\mathsf{first}(0), \mathsf{next}(0,1), \ldots, \mathsf{next}(i-1,i)\}$ for $i \geq 0$ witness that $\mathbb{P}$ has unbounded cardinality. ◀

▶ **Theorem 16.** *Let $k \in \mathbb{N}$ and let $\mathbb{P}$ be a* $\mathsf{Datalog}^{CV}$ *program. Deciding if $\mathbb{P}$ has k-bounded cardinality is* 2ExpTime*-complete, and* ExpTime*-complete for schemas of bounded height.*

Proof. Let $\star_0, \ldots, \star_k$ be distinct constants not occurring in $\mathbb{P}$. The *k-critial instance* $\mathbb{D}_k$ has, for every EDB predicate $p \in \mathbf{P}^{\text{EDB}}$, all possible facts constructed using $\star_0, \ldots, \star_k$: if p has sort Δ, then $\mathbb{D}_k$ contains facts $p(\star_0), \ldots, p(\star_k)$; if p has sort $\langle \Delta_1, \ldots, \Delta_\ell \rangle$, then $\mathbb{D}_k$ contains all facts $p(\star_{i_1}, \ldots, \star_{i_\ell})$ with $i_1, \ldots, i_\ell \in \{1, \ldots, k\}$.

Since entailment for $\mathsf{Datalog}^{\text{CV}}$ is monotonic (i.e., adding facts to a database only ever leads to more entailed facts), any fact entailed by $\mathbb{P}$ over a database containing $k+1$ distinct constants is also entailed over $\mathbb{D}_k$.

If all facts entailed by $\mathbb{P}$ over $\mathbb{D}_k$ are k-bounded, then $\mathbb{P}$ has k-bounded cardinality, otherwise we find at least one non-k-bounded fact witnessing that $\mathbb{P}$ does not have k-bounded cardinality. To compute these facts, we can use the procedure from Theorem 4.

The computation can be stopped as soon as a fact with a set of cardinality $> k$ is inferred. Hence, for analysing complexity, we only need to consider inferences of facts with smaller sets. We can estimate the number of terms that respect this restriction by viewing each term as a tree of depth bounded by the height h of the schema, and branching factor f bounded by the maximum of k and the arity of any tuple sort. Over $k+1$ constants in $\mathbb{D}_k$, there can be at most $(k+1)^{f^h}$ such terms. This translates into a double exponential bound for the number of derivable facts, and an according complexity bound as claimed. If the height of the schema is bounded, h can be considered constant and the same calculation leads to a single exponential bound (depending on f). ◀

Derivations of "large" sets involve either recursive rule applications to construct them from smaller sets, or unsafe set variables. Thus, the absence of either feature is a sufficient condition for bounded cardinality. This is similar to how *acyclicity notions* syntactically ensure chase termination on all input databases for sets of TGDs by restricting the propagation of nulls [15]. Indeed, we adapt Weak Acyclicity [18, 19] into a criterion for bounded cardinality.

▶ **Definition 17.** *We encode* positions *in sorts and terms as lists of natural numbers, where ε is the empty list and $\cdot$ is concatenation (which we generalise to sets of lists in the usual way). The set of positions* $\mathrm{Pos}(\tau)$ *of a sort τ, is defined recursively as follows:*

- $\mathrm{Pos}(\Delta) := \{\varepsilon\}$*;*
- $\mathrm{Pos}(\{\tau'\}) := \{\varepsilon\} \cup (1 \cdot \mathrm{Pos}(\tau'))$*; and*
- $\mathrm{Pos}(\langle \tau_1, \ldots, \tau_\ell \rangle) := \{\varepsilon\} \cup \bigcup_{i=1}^{\ell} (i \cdot \mathrm{Pos}(\tau_i))$.

Given a term t of sort τ, the subterms $\mathrm{At}(t, w)$ *at a position $w \in \mathrm{Pos}(\tau)$ are defined recursively as follows:*

- $\mathrm{At}(t, \varepsilon) := t$*;*
- $\mathrm{At}(\{t_1, \ldots, t_n\}, 1 \cdot v) := \bigcup_{i=1}^{n} \mathrm{At}(t_i, v)$*;*
- $\mathrm{At}(\langle t_1, \ldots, t_\ell \rangle, i \cdot v) := \mathrm{At}(t_i, v)$*;*

- $\mathrm{At}(t_1 \cup t_2, v) \coloneqq \mathrm{At}(t_1, v) \cup \mathrm{At}(t_2, v)$*; and*
- $\mathrm{At}(t_1 \cap t_2, v) \coloneqq \mathrm{At}(t_1, v) \cup \mathrm{At}(t_2, v)$,

where v is a (possibly empty) list of natural numbers. Note that $\mathrm{At}(\{\}, v \cdot 1) = \emptyset$.

All terms in $\mathrm{At}(t, w)$ *are of the same sort, which we call the* sort of t at position w. *A variable x* occurs at *position w in term t of sort τ if the sort of x is the sort of t at position w, and x occurs in a term in* $\mathrm{At}(t, w)$.

A predicate position *is a list $p \cdot w$ for a position $w \in \mathrm{Pos}(\mathsf{sort}(p))$; we denote by* $\mathrm{Pos}(p)$ *the set of all predicate positions for p. A variable x* occurs *at a predicate position $p \cdot w$ in a formula φ if φ contains an atom $p(t)$ and x occurs at position w in t.*

▶ **Example 18.** Consider the term $t = \langle a, \{\{x\}, S \cup \{y\}\}\rangle$ of sort $\tau = \langle \Delta, \{\{\Delta\}\}\rangle$. The positions of τ are $\mathrm{Pos}(\tau) = \{\varepsilon, 1, 2, 2 \cdot 1, 2 \cdot 1 \cdot 1\}$, and we have $\mathrm{At}(t, 1) = \{a\}$, $\mathrm{At}(t, 2) = \{\{\{x\}, S \cup \{y\}\}\}$, and $\mathrm{At}(t, 2 \cdot 1) = \{\{x\}, S \cup \{y\}\}$. We find that S occurs at $2 \cdot 1$ whereas x and y occur at $2 \cdot 1 \cdot 1$.

▶ **Definition 19.** *Consider a* $\mathsf{Datalog}^{CV}$ *program $\mathbb{P}$. The* WSA graph $G(\mathbb{P})$ *of $\mathbb{P}$ is a directed graph with two kinds of edges ("normal" and "special"). The vertices of $G(\mathbb{P})$ are all predicate positions of predicates in $\mathbb{P}$. $G(\mathbb{P})$ contains the following edges:*

1. *For every rule $\varphi \to \psi \in \mathbb{P}$, and every variable x that occurs in a body atom $q(s) \in \varphi$ at position $q \cdot v$ and in a head atom $p(t) \in \psi$ at position $p \cdot w$, there is an edge $q \cdot v \to p \cdot w$; it is* special *if x occurs in a subterm of the form $S \cup T$ in* $\mathrm{At}(t, w)$*, and* normal *otherwise.*
2. *For every rule $\varphi \to \psi \in \mathbb{P}$, and every variable x of set sort $\{\tau\}$ or tuple sort $\langle \tau_1, \ldots, \tau_\ell \rangle$, where a set sort occurs (directly or transitively) in a component sort τ_i, that occurs in a head atom $p(t) \in \psi$ at position $p \cdot w$ and that does not occur in the body φ, there is a special edge $p \cdot w \to p \cdot w$.*
3. *If there is an edge $q \cdot v \to p \cdot w$, and if the sort τ at position $q \cdot v$ (which is the same as the sort at position $p \cdot w$) has a position $u \in \mathrm{Pos}(\tau)$, then there is a normal edge $q \cdot v \cdot u \to p \cdot w \cdot u$.*

$\mathbb{P}$ *is* weakly set-acyclic *if $G(\mathbb{P})$ does not contain a directed cycle that involves a special edge.*

Intuitively, edges of the first kind model the propagation of values by rule applications. Edges of the second kind correspond to unsafe variables ranging over values containing sets. Such values always include sets that contain all constants occurring in the database. Thus, a program containing such a rule generally does not have bounded cardinality, and we thus always force a cycle in the WSA graph. Lastly, edges of the third kind model the propagation of values inside values of composite sorts. Indeed, weak set-acyclicity is easy to check:

▶ **Lemma 20.** *Deciding if $\mathbb{P}$ is weakly set-acyclic is* NL*-complete.*

Proof sketch. The WSA graph can be constructed by a logspace transducer, since the presence of any edge can be decided in L. Checking for cycles in a logspace-computable directed graph is NL-complete. ◀

We also note that any sufficiently large set produced by some program involves either an unsafe set variable, or the recursive application of some rule:

▶ **Lemma 21.** *For a* $\mathsf{Datalog}^{CV}$ *program $\mathbb{P}$ without unsafe set variables, there is $n \in \mathbb{N}$ such that a minimal chase sequence deriving a set S with $|S| \geq n$ has some rule ρ applied to a match containing a set T, where ρ is again part of the subsequence deriving T.*

Proof. Let $\rho \in \mathbb{P}$ be a rule and consider a match h for ρ. There is a polynomial $p(x)$ such that any set obtained by applying ρ yields a set of cardinality at most $p(x)$, provided that no set in h has cardinality exceeding x. Let $q(x)$ be a simultaneous upper bound for all such polynomials for rules in $\mathbb{P}$, consider the $|\mathbb{P}|$-fold composition $q^{|\mathbb{P}|} = q(\cdots(q(x))\cdots)$, and define $n := q^{|\mathbb{P}|}(1) + 1$.

If there is no database $\mathbb{D}$ such that $\mathbb{P}$ derives a set S of cardinality $|S| \geq n$ on $\mathbb{D}$, then $\mathbb{P}$ has n-bounded cardinality, and the claim is vacuously true. Otherwise, let S be such a set and $\mathbb{D}$ be such a database. Consider a minimal chase sequence deriving S. Any chase sequence without duplicate rules has at most $\mathbb{P}$ steps, and since $\mathbb{D}$ does not contain sets, such a sequence derives a set of cardinality at most $q^{|P|}(1) < n$. Thus, the minimal chase sequence deriving S must contain some rule ρ occurring in steps i and $j > i$, where step i derives a set T with $|T| < |S|$. ◀

We therefore find weak set-acyclicity a sufficient condition for bounded cardinality:

▶ **Theorem 22.** *If $\mathbb{P}$ is weakly set-acyclic, then $\mathbb{P}$ has bounded cardinality.*

Proof. Towards a contradiction, assume that $\mathbb{P}$ has unbounded cardinality, but $G(\mathbb{P})$ does not contain a directed cycle traversing a special edge. Let n be the constant from Lemma 21. Since $\mathbb{P}$ has unbounded cardinality, there is a set S with $|S| \geq n$ that $\mathbb{P}$ derives, consider a minimal chase sequence for some fact containing it. Note that any rule application in this sequence corresponds to at least one edge in $G(\mathbb{P})$. By Lemma 21, there is a set T derived by a rule ρ that is propagated to some fact partaking in another match for ρ. Thus, ρ is part of a directed cycle in $G(\mathbb{P})$. Furthermore $|T| < |S|$, since the chase sequence is minimal. Thus one of the rule applications corresponding to this cycle must involve a union in the head of the rule, making the corresponding edge in $G(\mathbb{P})$ special. ◀

Weakly set-acyclic programs constitute a tractable fragment of $\mathsf{Datalog}^{\mathrm{CV}}$ for which membership is decidable. But even simple bounded-cardinality programs may not be WSA:

▶ **Example 23.** Consider again the programs $\mathbb{P}$ and $\mathbb{P}'$ from Example 12. The WSA graph of $\mathbb{P}$ has edges $e \cdot \epsilon \to s \cdot 1$, $s \cdot \epsilon \to p \cdot \epsilon$, and $s \cdot 1 \to p \cdot 1$. Therefore, $\mathbb{P}$ is WSA. The WSA graph for $\mathbb{P}'$ contains the special edge $s \cdot \epsilon \to s \cdot \epsilon$, and $\mathbb{P}'$ is not WSA. Let $\bar{\mathbb{P}}$ consist of $\mathbb{P}$ and (44):

$$s(X) \wedge s(Y) \wedge p(S) \to s(S \cap (X \cup Y)) \tag{44}$$

The WSA graph of $\bar{\mathbb{P}}$ has the special edge $s \cdot \epsilon \to s \cdot \epsilon$, but $\bar{\mathbb{P}}$ has 2-bounded cardinality.

We can improve over WSA by estimating the maximal cardinality of sets produced by a rule more cautiously. We therefore construct a system of inequalities that correspond to lower bounds on the cardinalities of sets. These bounds are expressions built from natural numbers and the operators $+$, min, and max. Then the sum of these lower bounds has a minimum if and only if the program has bounded cardinality.

▶ **Definition 24.** *Let $\mathbb{P}$ be a $\mathsf{Datalog}^{CV}$ program. For a set sort $\tau = \{\tau'\}$, let P_τ be the set of all predicate positions of sort τ. A predicate position $p \cdot w$ in P_τ is a* target position *if p occurs in the head of some rule $\rho \in \mathbb{P}$. With every predicate position $p \cdot w \in P_\tau$, associate a variable $x_{p \cdot w}$. Set $X := \bigcup_{\tau = \{\tau'\}} \{x_{p \cdot w} \mid p \cdot w \in P_\tau\}$, and let $T \subseteq X$ be the set of all target variables. Recursively define the* lower bound $[\![t]\!]^\rho_{p \cdot w}$ *of a term t occurring at $p \cdot w$ in rule ρ:*

1. *if $t \in \mathbf{V}$ is a variable, let* $\mathrm{bpos}(t)$ *be the set of all predicate positions of* $\mathrm{body}\,\rho$ *that t occurs at and set $[\![t]\!]^\rho_{p \cdot w} := \max\{x_{q \cdot v} \mid q \cdot v \in \mathrm{bpos}(t)\}$;*
2. *$[\![\{t_1, \ldots, t_\ell\}]\!]^\rho_{p \cdot w} := \ell$;*

3. $[\![(t_1 \cap t_2)]\!]^{\rho}_{p\cdot w} \coloneqq \min([\![t_1]\!]^{\rho}_{p\cdot w}, [\![t_2]\!]^{\rho}_{p\cdot w})$; *and*
4. $[\![(t_1 \cup t_2)]\!]^{\rho}_{p\cdot w} \coloneqq [\![t_1]\!]^{\rho}_{p\cdot w} + [\![t_2]\!]^{\rho}_{p\cdot w}$.

Every variable $x \in X$ has the associated inequality $x \geq 0$, every target variable $x_{p\cdot w}$ has associated inequalities for terms t occurring at $p \cdot w$ in the head of rule ρ: $x_{p\cdot w} \geq [\![t]\!]^{\rho}_{p\cdot w}$. The system of cardinality constraints $\mathsf{card}(\mathbb{P})$ *is the system of all such inequalities. The* cardinality constraints problem *is minimising $\sum_{x\in T} x$ subject to* $\mathsf{card}(\mathbb{P})$.

Intuitively speaking, we obtain a bound ℓ for set literals $\{t_1, \ldots, t_\ell\}$ by assuming that all terms are distinct. For intersections $t_1 \cap t_2$, the bound is the minimum of the bounds for t_1 and t_2, whereas for unions, it is the sum of the bounds (assuming that, in the worst case, the sets are disjoint). For variables, we simply take the maximum over all bounds for the same sort that occur in the body of the rule.

The cardinality constraints problem allows us to strengthen the bound from Lemma 21:

▶ **Lemma 25.** *Let $\mathbb{P}$ be a* $\mathsf{Datalog}^{CV}$ *program. If the cardinality constraints problem has optimal value k, any minimal chase sequence deriving a set S with $|S| \geq k+1$ has some rule ρ applied to a match containing a set T, where ρ is again part of the subsequence deriving T.*

▶ **Theorem 26.** *If the cardinality constraints problem for some* $\mathsf{Datalog}^{CV}$ *program $\mathbb{P}$ has an optimal solution, then $\mathbb{P}$ has bounded cardinality. Deciding whether this is the case can be done in polynomial time with respect to the size of $\mathbb{P}$.*

Proof. Note that the cardinality constraints problem for $\mathbb{P}$ is always *bounded*, i.e., it always has only finitely many possible solutions, since we require $x \geq 0$ for all variables $x \in X$. Thus, it has an optimal solution precisely when it has at least one solution, i.e., when it is *feasible*. Assume for a contradiction that $\mathbb{P}$ has unbounded cardinality, but that the cardinality constraints problem has optimal value k. By Lemma 25, there is a set S with $|S| \geq k+1$ such that any minimal chase sequence applies rule ρ to some set T with ρ part of the subsequence deriving T. Without loss of generality, assume that both S and T occur at the same predicate position $p \cdot w$ (since $\mathbb{P}$ has unbounded cardinality, we can chose S large enough) in steps i and $j > i$. Let $p \cdot w = q^i \cdot v^i, q^{i+1} \cdot v^{i+1}, \ldots, q^{j-1} \cdot v^{j-1}, q^j \cdot v^j = p \cdot w$ be the positions along the subsequence. We have $x_{p\cdot w} \geq x_{q^{i+1}\cdot v^{i+1}} \geq \cdots \geq x_{q^{j-1}\cdot v^{j-1}} \geq x_{p\cdot w}$. Since $|S| > |T|$, one of the rules applied in the subsequence involves a union. Thus one of the inequalities is strict and the cardinality constraints problem is infeasible, which is the desired contradiction.

Since $\mathbb{P}$ only has polynomially many positions, the cardinality constraints problem for $\mathbb{P}$ is of polynomial size with respect to $\mathbb{P}$. By using additional variables, $\mathsf{card}(\mathbb{P})$ can be transformed into an equifeasible linear program (i.e., a system of linear inequalities that admits an optimal solution precisely when $\mathsf{card}(\mathbb{P})$ does): inequalities of the form $x \geq \max\{t_1, \ldots, t_\ell\}$ are replaced by ℓ inequalities $x \geq t_1, \ldots, x \geq t_\ell$, and inequalities of the form $x \geq \min(t_1, t_2)$ are replaced by $x = t_1 - y$, $y \geq 0$, and $y \geq t_1 - t_2$, where y is a fresh variable. The resulting linear program is still of polynomial size, and solving such linear programs can be done in polynomial time [24]. ◀

The system of cardinality constraints is not a sufficient condition for bounded cardinality either, but it can capture bounded cardinality for programs that are not weakly set-acyclic:

▶ **Example 27.** Consider again the program $\bar{\mathbb{P}}$ from Example 23. Then $\mathsf{card}(\bar{\mathbb{P}})$ consists of the following inequalities with optimal solution $x_{s\cdot\epsilon} = 2 = x_{p\cdot\epsilon}$ and optimal value $k = 4$:

$$x_{s\cdot\epsilon} \geq 0 \qquad x_{p\cdot\epsilon} \geq 0$$
$$x_{s\cdot\epsilon} \geq 1 \qquad x_{p\cdot\epsilon} \geq 2 \qquad x_{s\cdot\epsilon} \geq \min(x_{p\cdot\epsilon}, x_{s\cdot\epsilon} + x_{s\cdot\epsilon})$$

6 Linear Datalog$^{\text{CV}}$

For both Datalog and TGDs, linearity is a syntactic criterion that corresponds to a fragment with lower complexities [21, 20]. It thus seems prudent to investigate whether the analogous fragment of Datalog$^{\text{CV}}$ enjoys similarly reduced complexities.

▶ **Definition 28.** *A* Datalog^{CV} *rule* $\varphi \to \psi$ *is a* linear Datalog^{CV} *rule if both* φ *and* ψ *each comprise exactly one atom. A* Datalog^{CV} *program* $\mathbb{P}$ *is linear if all rules* $\rho \in \mathbb{P}$ *are linear.*

Unlike for Datalog$^{\text{CV}}$, we allow databases for linear Datalog$^{\text{CV}}$ programs to contain facts of arbitrary sorts, as such facts cannot generally be constructed using linear rules. We find that with one added layer of nested sets, deciding entailment for linear Datalog$^{\text{CV}}$ is just as hard as for Datalog$^{\text{CV}}$. In other words, for the same set height, the complexity of linear Datalog$^{\text{CV}}$ is indeed at most one exponent lower than for non-linear Datalog$^{\text{CV}}$.

▶ **Theorem 29.** *Let* $\mathbf{S}$ *be a schema with* $k = \mathsf{s\text{-}height}(\mathbf{S}) > 0$*, let* $\mathbb{P}$ *be a linear* Datalog^{CV} *program and* $\mathbb{D}$ *a database over* $\mathbf{S}$*, and let* α *be a ground fact. Deciding* $\mathbb{P}, \mathbb{D} \models \alpha$ *is* $(k-1)$ExpTime*-hard for data complexity and* $(k+1)$ExpTime*-hard for combined complexity. If* $\mathsf{t\text{-}height}(\mathbf{S})$ *is bounded, the combined complexity drops to* kExpTime*-hard.*

Proof sketch. The proof follows the idea from the proof of Theorem 6. We use a single predicate facts holding tuples of sets, with each component corresponding to one of the original predicates: Let $p_1, p_2, \dots$ be a fixed enumeration of all predicates occurring in $\mathbb{P}$ with sorts $\tau_1, \tau_2, \dots$. Then facts has sort $\langle \{\tau_1\}, \{\tau_2\}, \dots \rangle$, and component i corresponds to facts for p_i. For a formula Φ, define $\mathrm{ts}_\Phi(p) := \{\boldsymbol{t}\}$, where $\boldsymbol{t}$ is the (possibly empty) list of terms t for which $p(t)$ occurs in Φ. We then translate a rule $\rho = \varphi \to \psi$ into (45), and translate $\mathbb{D}$ as $\mathsf{dbFacts}(\mathrm{ts}_{\mathbb{D}}(p_1), \mathrm{ts}_{\mathbb{D}}(p_2), \dots)$, where we view $\mathbb{D}$ as a conjunction of atoms.

$$\mathsf{facts}(y_1 \cup \mathrm{ts}_\varphi(p_1), y_2 \cup \mathrm{ts}_\varphi(p_2), \dots) \to \mathsf{facts}(y_1 \cup \mathrm{ts}_\rho(p_1), y_2 \cup \mathrm{ts}_\rho(p_2), \dots) \tag{45}$$

Lastly, the linear rule $\mathsf{dbFacts}(\boldsymbol{y}) \to \mathsf{facts}(\boldsymbol{y})$ derives the initial facts from $\mathbb{D}$. ◀

Note that, even with this encoding, negation cannot be simulated by $\notin$, since we cannot require rules to be applied only after all facts for some predicate p_i have been derived.

7 Discussion and Future Work

We have formalised Datalog$^{\text{CV}}$, a positive extension of Datalog with complex values, identified its complexity, and developed a translation into terminating TGD sets. In sharp contrast to Relationlog [28], which has the same complexity as Datalog, Datalog$^{\text{CV}}$ can express highly complex queries, and unlike Set-extended Datalog [39], it supports tuples as well as sets.

We have shown that bounded cardinality programs form a fragment with tractable reasoning. Since it is undecidable whether a program has bounded cardinality, we have proposed two decidable sufficient conditions for bounded cardinality: weak set-acyclicity and the cardinality constraints problem.

Regarding future works, our translations put complex value reasoning in reach of rule engines such as VLog [11] and RDFox [31], which have the potential of addressing many of the applications from the introduction. On the theoretical side, *(stratified) negation* would be a natural extension to study, also regarding its utility for expressing query languages such as MARPL [30] and eMARPL [29]. Moreover, while it was recently shown that chase-terminating tuple-generating dependencies capture all decidable monotonic queries [8], the expressive power of Datalog$^{\text{CV}}$ and its tractable fragments remains open.

References

1 Serge Abiteboul and Stéphane Grumbach. COL: A logic-based language for complex objects. In Joachim W. Schmidt, Stefano Ceri, and Michele Missikoff, editors, *Proc. 1st Int. Conf. on Extending Database Technology (EDBT'88)*, volume 303 of *LNCS*, pages 271–293. Springer, 1988.

2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1994.

3 Serge Abiteboul and Paris C. Kanellakis. Object identity as a query language primitive. *J. ACM*, 45(5):798–842, 1998.

4 Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. Rewriting guarded existential rules into small datalog programs. In Benny Kimelfeld and Yael Amsterdamer, editors, *Proc. 21st Int. Conf. on Database Theory (ICDT'18)*, volume 98 of *LIPIcs*, pages 4:1–4:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

5 Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutiérrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. G-CORE: A core for future graph query languages. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proc. 2018 ACM SIGMOD Int. Conf. on Management of Data*, pages 1421–1432. ACM, 2018.

6 Catriel Beeri, Shamim A. Naqvi, Raghu Ramakrishnan, Oded Shmueli, and Shalom Tsur. Sets and negation in a logic database language (LDL1). In Moshe Y. Vardi, editor, *Proc. 6th Symposium on Principles of Database Systems (PODS'87)*, pages 21–37. ACM, 1987.

7 Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. of the ACM*, 31(4):718–741, 1984.

8 Camille Bourgaux, David Carral, Markus Krötzsch, Sebastian Rudolph, and Michaël Thomazo. Capturing homomorphism-closed decidable queries with existential rules. In Meghyn Bienvenu, Gerhard Lakemeyer, and Esra Erdem, editors, *Proc. 18th International Conference on Principles of Knowledge Representation and Reasoning (KR'21)*, pages 141–150, 2021.

9 Pierre Bourhis, Juan L. Reutter, and Domagoj Vrgoc. JSON: data model and query languages. *Inf. Syst.*, 89:101478, 2020.

10 Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Finitely recursive programs: Decidability and bottom-up computation. *J. of AI Commun.*, 24(4):311–334, 2011.

11 David Carral, Irina Dragoste, Larry González, Ceriel J. H. Jacobs, Markus Krötzsch, and Jacopo Urbani. Vlog: A rule engine for knowledge graphs. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *Proc. 18th Int. Semantic Web Conf. (ISWC'19), Part II*, volume 11779 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2019.

12 David Carral, Irina Dragoste, and Markus Krötzsch. Reasoner = logical calculus + rule engine. *KI*, 2020.

13 David Carral, Irina Dragoste, Markus Krötzsch, and Christian Lewe. Chasing sets: How to use existential rules for expressive reasoning. In Sarit Kraus, editor, *Proc. 28th Int. Joint Conf. on Artificial Intelligence (IJCAI'19)*. ijcai.org, 2019.

14 Angelos Charalambidis, Christos Nomikos, and Panos Rondogiannis. The expressive power of higher-order datalog. *Theory Pract. Log. Program.*, 19(5-6):925–940, 2019.

15 Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *J. of Artificial Intelligence Research*, 47:741–808, 2013.

16 Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

17 Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In Maurizio Lenzerini and Domenico Lembo, editors, *Proc. 27th Symposium on Principles of Database Systems (PODS'08)*, pages 149–158. ACM, 2008.

18 Alin Deutsch and Val Tannen. Reformulation of XML queries and constraints. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *Proc. 9th Int. Conf. on Database Theory (ICDT'03)*, volume 2572 of *LNCS*, pages 225–241. Springer, 2003.

19 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

20 Georg Gottlob, Marco Manna, and Andreas Pieris. Polynomial rewritings for linear existential rules. In Qiang Yang and Michael Wooldridge, editors, *Proc. 24th Int. Joint Conf. on Artificial Intelligence (IJCAI'15)*, pages 2992–2998. AAAI Press, 2015.

21 Georg Gottlob and Christos H. Papadimitriou. On the complexity of single-rule datalog queries. *Inf. Comput.*, 183(1):104–122, 2003.

22 Marc Gyssens, Dan Suciu, and Dirk Van Gucht. Equivalence and normal forms for the restricted and bounded fixpoint in the nested algebra. *Inf. Comput.*, 164(1):85–117, 2001.

23 Jan Hidders, Jan Paredaens, and Jan Van den Bussche. J-logic: Logical foundations for JSON querying. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proc. 36th Symposium on Principles of Database Systems (PODS'17)*, pages 137–149. ACM, 2017.

24 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

25 Michael Kifer and James Wu. A logic for programming with complex objects. *J. of Comput. Syst. Sci.*, 47(1):77–120, 1993.

26 Markus Krötzsch, Maximilian Marx, and Sebastian Rudolph. The power of the terminating chase. In *Proc. 22nd Int. Conf. on Database Theory (ICDT'19)*, volume 127 of *LIPIcs*, pages 3:1–3:17. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019.

27 Gabriel M. Kuper. Logic programming with sets. *J. of Comput. Syst. Sci.*, 41(1):44–64, 1990.

28 Mengchi Liu. Relationlog: A typed extension to datalog with sets and tuples. *J. of Log. Program.*, 36(3):271–299, 1998.

29 David Martin and Peter F. Patel-Schneider. Wikidata constraints on MARS. In Lucie-Aimée Kaffee, Oana Tifrea-Marciuska, Elena Simperl, and Denny Vrandečić, editors, *Proc. 1st Wikidata Workshop (Wikidata 2020)*, volume 2773 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

30 Maximilian Marx, Markus Krötzsch, and Veronika Thost. Logic on MARS: Ontologies for generalised property graphs. In Carles Sierra, editor, *Proc. 26th Int. Joint Conf. on Artificial Intelligence (IJCAI'17)*, pages 1188–1194. ijcai.org, 2017.

31 Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. Rdfox: A highly-scalable RDF store. In Marcelo Arenas, Óscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab, editors, *Proc. 14th Int. Semantic Web Conf. (ISWC'15), Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2015.

32 Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10)*, pages 269–279. AAAI Press, 2010.

33 Jan Paredaens and Dirk Van Gucht. Converting nested algebra expressions into flat algebra expressions. *ACM Trans. Database Syst.*, 17(1):65–93, 1992.

34 Marko A. Rodriguez and Peter Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010.

35 Michael Rudolf, Marcus Paradies, Christof Bornhövd, and Wolfgang Lehner. The graph story of the SAP HANA database. In *Datenbanksysteme für Business, Technologie und Web (BTW), 15. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 11.-15.3.2013 in Magdeburg, Germany. Proceedings*, pages 403–420, 2013.

36 Dan Suciu. Bounded fixpoints for complex objects. *Theor. Comput. Sci.*, 176(1-2):283–328, 1997.

37 Shalom Tsur and Carlo Zaniolo. LDL: A logic-based data language. In Wesley W. Chu, Georges Gardarin, Setsuo Ohsuga, and Yahiko Kambayashi, editors, *Proc. 12th Int. Conf. on Very Large Data Bases (VLDB'86)*, pages 33–41. Morgan Kaufmann, 1986.

38 Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10), 2014.

39 Qing Zhou and Ligong Long. SEDatalog: A set extension of datalog. In Zhongzhi Shi and Qing He, editors, *Proc. 2nd Int. Conf. on Intelligent Information Processing (IFIP'04)*, volume 163 of *IFIP*, pages 383–388. Springer, 2004.

Inference of Shape Graphs for Graph Databases

Benoît Groz
University Paris Sud, France

Aurélien Lemay
University of Lille, France

Sławek Staworko
University of Lille, France

Piotr Wieczorek
University of Wrocław, Poland

Abstract

We investigate the problem of constructing a shape graph that describes the structure of a given graph database. We employ the framework of *grammatical inference*, where the objective is to find an inference algorithm that is both *sound*, i.e., always producing a schema that validates the input graph, and *complete*, i.e., able to produce any schema, within a given class of schemas, provided that a sufficiently informative input graph is presented. We identify a number of fundamental limitations that preclude feasible inference. We present inference algorithms based on natural approaches that allow to infer schemas that we argue to be of practical importance.

2012 ACM Subject Classification Information systems → Graph-based database models

Keywords and phrases RDF, Schema, Inference, Learning, Fitting, Minimality, Containment

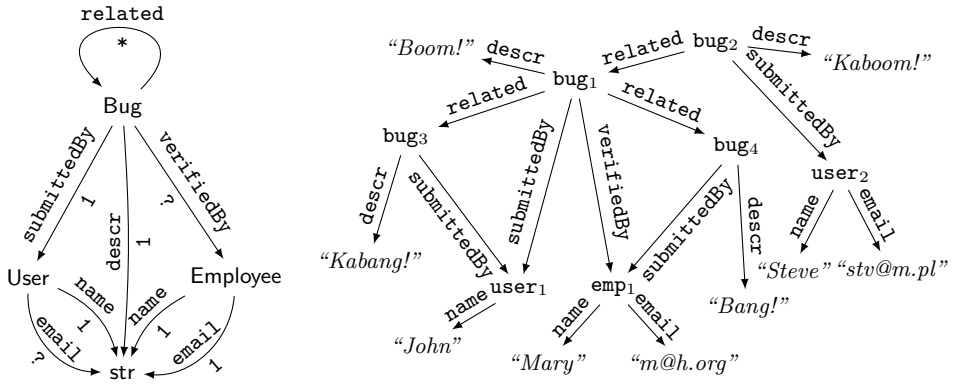
Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.14

Funding *Piotr Wieczorek*: This author has been partially supported by Polish National Science Center grant NCN 2016/23/B/ST6/01438.

1 Introduction

Traditionally, in relational databases, defining the schema is the mandatory first step before a database can even be populated with data. However, novel database models, such as graph databases, quite intentionally allow to store and process data without declaring any schema. This facilitates the evolution of the structure of a database while the applications around it are being developed. In fact, often a suitable schema formalism is proposed after a particular database model has established its place in practice. In those circumstances a natural problem of *schema inference* arises: given a schema-less database construct a schema that captures the structure of the database. This problem has been identified as an important research direction [1] and is well motivated since the knowledge of database structure is instrumental in any meaningful data processing task such as querying or transformation.

In the present paper, we present a principled approach to the problem of schema inference for graph databases. We consider RDF graphs and Shape Expression Schemas (ShEx) [55, 47]. ShEx builds on the success of XML Schema and allows to describe the *structure* of an RDF graph by defining patterns of arrangement of RDF nodes. More precisely, ShEx specifies a collection of node types, each type defined by a regular expression that constrains the types of the outbound neighborhood of a node. Take for instance the RDF graph storing bug reports, presented in Figure 1 together with its shape expression schema. The schema requires a bug report to have a description and a user who submitted it. Optionally, a bug report may have an employee who verified it. Also, a bug report can have a number of related bug reports. A user has a name and an optional email address while an employee has a name and a mandatory email address. We point out that just like with XML Schema the



Bug → `descr` : str, `submittedBy` : User, `verifiedBy` : Employee?, `related` : Bug*
User → `name` : str, `email` : str?
Employee → `name` : str, `email` : str

Figure 1 An RDF graph with bug reports (top right) together with a shape expression schema (bottom) and the corresponding shape graph (top left). str is a built-in type for literal string nodes.

nodes of the RDF graph need not be typed and it is the task of a validation algorithm to find a valid node typing [42, 35, 47]. Furthermore, some nodes may need to be assigned more than one type, for instance the node $\texttt{emp}_1$ needs to satisfy the types User and Employee.

We focus our investigation on a practical subclass ShEx_0 that defines types with collections of atoms with *arities* ranging over `1`, `?`, `+`, and `*`. This fragment does not allow disjunction or union types but can adequately approximate schemas with those features. ShEx_0 is particularly suited to capture the topology of RDF graphs obtained by exporting relational databases in a number of formalisms proposed for this task, such as R2RML, Direct Mapping, and YARRRML [46, 45, 13]. More importantly, the class ShEx_0 enjoys a sought-after feature of an equivalent graphical representation in the form of a *shape graph* whose nodes are types and edges are labeled by a symbol and a multiplicity (see Figure 1).

We present a principled approach to the problem of graph database schema inference, where we start with the fundamental question: *What is an inference algorithm?*. We answer it using the framework of *grammatical inference* [28], which in recent years has been successfully applied to a number of database formalisms ranging from queries [17, 48] to schemas [7, 22] and transformations [38, 36]. In essence, an inference algorithm needs to be both *sound*, i.e., produce a schema that validates the input graph, and *complete*, i.e., able to infer any goal schema with a sufficiently informative input graph. This formal framework allows us to tackle another fundamental question: *When is inference feasible?* Indeed, we identify classes of schemas that are not learnable, which reveals two principal challenges in schema inference: 1) distinguishing unbounded arities `*` and `+` from bounded ones `1` and `?`, and 2) distinguishing recursive types from their finite non-recursive unravelings.

A suitable ShEx_0 inference algorithm needs to identify the types and their definitions. ShEx_0 assigns to every node of a graph a set of types by using the notion of *embedding*, which is an extension of the standard graph simulation. Consequently, our first attempt is to use graph simulation to identify sets of nodes whose *outbound* neighborhood shares structural similarity, and therefore, should have the same type. Interestingly, this approach yields a very reasonable inference algorithm that generalizes the structural information of the input graph, and in particular, easily introduces recursion in the constructed schema. The generalization is however very eager, and in particular, the algorithm clumps together any two types that are related by subtype relation such as Employee $\subseteq$ User where every node of

type Employee has also type User. Indeed, this inference algorithm produces a shape that is *singular*, having no two types that cannot be distinguished with the help of simulation, and consequently it prohibits schemas with subtypes like the schema in Figure 1.

To address this shortcoming we investigate using the *context* in which nodes are used: their *inbound* neighborhood. Indeed, the nodes of type User and the nodes of type Employee are used in different contexts, although their use may partially overlap e.g., the node $\texttt{emp}_1$ is used both as an employee and as a user. An inference algorithm based on using context information constructs a schema that belongs to the class of *deterministic* shape graph, which permits every edge label to be present at most once in a type definition. This class of schemas is incomparable with singular shape graph. For instance, the schema in Figure 1 is deterministic but is not singular. We find, however, that an indiscriminate use of the context information leads to *overfitting*, and in particular, it may fail to introduce recursion and may produce voluminous schemas. It appears that an inference algorithm must choose carefully the information to identify the set of types. We propose, as a proof of concept, a hybrid inference algorithm that extends the simulation-based approach with a modest amount of context information that consists of a single incoming edge. This algorithm produces schemas that are *context-singular*, a generalization of singular schemas that allows subtyping between types used in different contexts.

Finally, we survey a number of graph databases and inspect their schemas. Our analysis shows that ShEx_0 and the proposed subclasses lack certain features present in real-world schemas, most notably union types, but we can approximate them very well. This makes our algorithms an excellent solution for preparing the first draft of a schema to be refined by a knowledgeable architect.

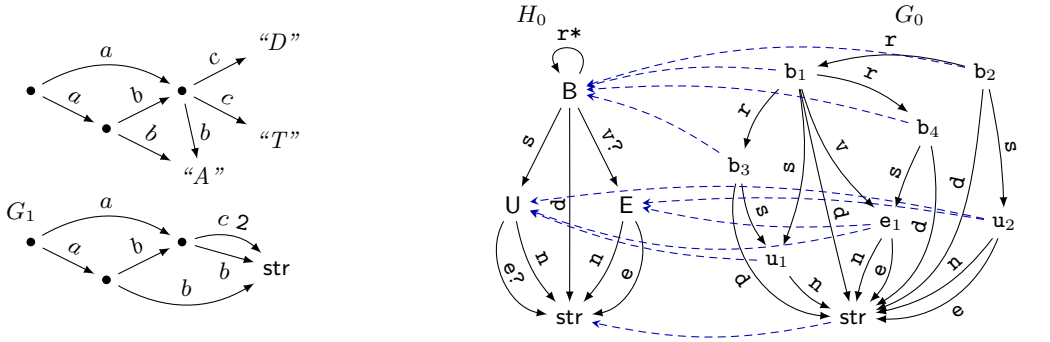
Organization. The paper is organized as follows. In Section 2 we present basic notions, including shape graphs and their embeddings. In Section 3 we formally state the problem of inference. In Section 4 we present a simulation-based inference algorithm for the class of singular shape graphs. In Section 5 we present negative implications of unrestrained use of context information in inference. In Section 6 we propose a hybrid approach combining the simulation-based approach with elements of context information. In Section 7 we present an analysis of schemas of a number of graph databases. We discuss related work in Section 8. We summarize our findings and outline future research directions in Section 9.

2 Basic notions

Throughout this paper we apply a functional notation to relations, and conversely, often view functions as relations. For instance, for a binary relation $R \subseteq A \times B$ we set $\mathsf{dom}(R) = \{a \in A \mid \exists b \in B.\ (a,b) \in R\}$, $\mathsf{ran}(R) = \{b \in B \mid \exists a \in A.\ (a,b) \in R\}$, $R(a) = \{b \in B \mid (a,b) \in R\}$ for $a \in A$, and $R^{-1}(b) = \{a \in A \mid (a,b) \in R\}$ for $b \in B$.

Intervals. We use the standard notation $[i;j]$ for $0 \leq i \leq j \leq \infty$ to denote *intervals* that represent nonempty sets of consecutive natural numbers. By I_0 we denote the set of all intervals. Our schema formalisms use only four *basic intervals* (M_0) given with their shorthand notation: 1 is $[1;1]$, ? is $[0;1]$, + is $[1;\infty]$ and * is $[0;\infty]$. A *simple interval* has the form $[i;i]$ for $0 \leq i < \infty$, and often we use i for $[i;i]$. Throughout this paper, we employ the point-wise addition operation $[i_1;j_1] \oplus [i_2;j_2] = [i_1+i_2; j_1+i_2]$. A set of intervals M is *proper* if any interval is contained in an interval in M. For a proper M we define the function fit_M that maps any finite nonempty set of natural numbers (occurrences) into a smallest interval in M containing it.

Graphs and schemas. We employ a generic model of a graph that we use to model both RDF and the subclass ShEx_0 that allows type definitions consisting of collections of atoms using basic intervals [47, 49]. Our methods do not inspect data values of literal nodes, and consequently, we will abstract them using their type alone but this requires us to record the number of some types of edges as illustrated in Figure 2a.



(a) An RDF graph (above) and its abstraction as a simple graph (below).

(b) An embedding $G_0 \preccurlyeq H_0$ (concise versions of graphs from Figure 1).

Figure 2 Graphs and embeddings.

When drawing graphs we typically omit the label 1 and simple intervals are denoted with natural numbers e.g., 2 stands for $[2;2]$ which indicates two identical edges. We point out a dichotomy of node kinds and their types: literal nodes can only have literal types and non-literal nodes can only have non-literal types. While it might be tempting to remove literal types from consideration and focus on data-free graphs, we keep literal types and nodes because without them it is difficult to define reasonable schema families without subtypes.

We assume a finite set Σ of edge labels and an enumerable set of node identifiers $\mathcal{N} = \mathcal{U} \cup \mathcal{L}$, where $\mathcal{U}$ is an infinite set of identifiers used to represent both URIs and blank nodes, and $\mathcal{L}$ is a finite set of types of literal nodes e.g., str representing strings. We assume that $\mathcal{U} \cap \mathcal{L} = \emptyset$.

▶ **Definition 1.** *A* graph G *is a triple* $(N_G, E_G, \mathsf{arity}_G)$*, where* $N_G \subseteq \mathcal{N}$ *is a finite set of nodes,* $E_G \subseteq (N_G \cap \mathcal{U}) \times \Sigma \times N_G$ *is a set of labeled oriented edges, and* arity_G *maps every edge to an interval. A* shape graph *is a graph that uses only basic intervals and by* ShEx_0 *we denote the set of all shape graphs. A* simple graph *is a graph that assigns to every edge a simple interval and this simple interval is* 1 *unless the edge leads to a node in* $\mathcal{L}$*. By* G_0 *we denote the set of simple graphs. A simple graph* G *is* data-free *if* $N_G \cap \mathcal{L} = \emptyset$*.* ⌟

In Figure 2b we present a more concise version of the graphs from Figure 1 adapted to our graph model. For $M \subseteq \mathsf{M}_0$, we write $\mathsf{ShEx}_0(M)$ to restrict the choice of arities to M. For an edge $e = (n, p, m)$ we let $\mathsf{source}(e) = n$, $\mathsf{lab}(e) = p$, and $\mathsf{target}(e) = m$. For a given node n of G we identify its outbound neighborhood $\mathsf{out}_G(n) = \{e \in E_G \mid \mathsf{source}(e) = n\}$. Given two graphs G_1 and G_2 by $G_1 \uplus G_2$ we denote their disjoint union. In the sequel, we often refer to shape graphs as *schemas* and their nodes as *types*. We also abuse the standard tree terminology and say that nodes n and n' are siblings if there is a node m that has outgoing edges to n and to n'.

Embeddings. We recall next, and illustrate in Figure 2b, the notion of embedding that allows to define the semantics of shape graphs as schemas [49] and is instrumental in our inference methods. Here, by id_A we denote the identity relation on a given set A.

▶ **Definition 2** (Embedding [49]). *An* embedding *of a graph G in a graph H is a binary relation $R \subseteq (N_G \times N_H) \cap (\mathcal{U} \times \mathcal{U} \cup \mathsf{id}_{\mathcal{L}})$ such that for any $(n, m) \in R$ there exists a* witness *of embedding of n in m w.r.t. R, i.e., a function $\lambda : \mathsf{out}_G(n) \to \mathsf{out}_H(m)$ such that*

1. *for every $e \in \mathsf{out}_G(n)$ we have that $\mathsf{lab}(e) = \mathsf{lab}(\lambda(e))$,*
2. *for every $f \in \mathsf{out}_H(m)$ we have that $\bigoplus\{\mathsf{arity}_G(e) \mid \lambda(e) = f\} \subseteq \mathsf{arity}_H(f)$, where $\bigoplus$ stands for the point-wise sum of all the intervals.*
3. *for every $e \in \mathsf{out}_G(n)$ we have that $(\mathsf{target}(e), \mathsf{target}(\lambda(e))) \in R$.*

The embedding R is total *if $\mathsf{dom}(R) = N_G$. We write $G \preccurlyeq H$ if there exists a total embedding of G in H. An* autoembedding *of G is an embedding of G in itself.* ⌟

Figure 2b presents an example of the embedding between the concise versions of RDF graph and the shape graph in Figure 1. It is known that there exists exactly one *maximal embedding* between two graphs [47] (since the union of embeddings is also an embedding), and constructing maximal embeddings between pairs of shape graphs and between simple graphs and shape graphs is in P [49]. In the sequel, by $\ll_{G:H}$ we denote the maximal embedding of G in H, by $\ll_G$ the maximal autoembedding of G, and we use both relation symbols using infix notation. The *language* of a shape graph H is $L(H) = \{G \in \mathsf{G_0} \mid G \preccurlyeq H\}$. We recall from [49], that embeddings can be composed, and in particular, $G \preccurlyeq H$ implies $L(G) \subseteq L(H)$, but the converse does not necessarily hold as illustrated in Figure 3.

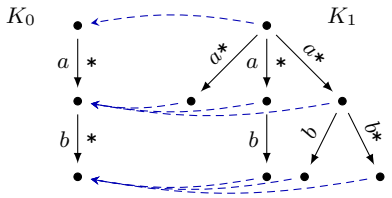


Figure 3 Containment does not imply an embedding: K_0 and K_1 are equivalent but $K_1 \preccurlyeq K_0$ and $K_0 \not\preccurlyeq K_1$.

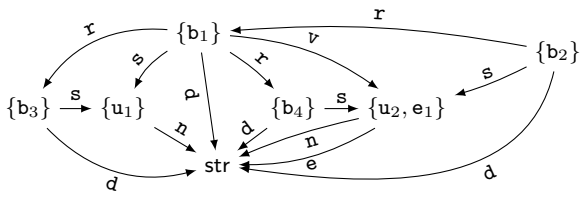


Figure 4 Aggregate of G_0 by grouping P_0 induced by the bisimulation relation of G_0.

Throughout the paper we use the following terminology. A simple graph G *satisfies* a shape graph H, and H *recognizes* G, iff $G \in L(H)$. Two shape graphs H and K are *equivalent*, in symbols $H \equiv K$, iff $L(H) = L(K)$. A node n of a simple graph G *has type* t of a shape graph H iff $n \ll_{G:H} t$. Given two types t_1 and t_2 of a shape graph H, a type t_1 is a *subtype* of t_2, in symbols $t_1 \subseteq_H t_2$, iff $\forall G \in L(H).\ \forall n \in N_G.\ n \ll_{G:H} t_1 \Rightarrow n \ll_{G:H} t_2$. Finally, a type t of H is *recursive* iff it is part of a loop in H.

3 Inference of shape graphs

In this section we introduce the framework of grammatical inference, present a number of limit point arguments that help to identify main challenges in inferring shape graphs, and finally, we present the construction of graph aggregation that we use to infer shape graphs.

3.1 Grammatical inference

In this paper we investigate the problem of *shape graph inference*, which consists of constructing a shape graph for a given input simple graph. To this end we adopt the framework of *grammatical inference* [28], which has originally been proposed for word languages and has been successfully applied to a number of database formalisms ranging

from queries [17, 48] to schemas [7, 22] to transformations [38, 36]. In essence, the framework requires the inference algorithm to be capable of inferring any goal schema given a sufficiently informative input. We are interested in inference from positive data: the input consists of a graph that belongs to the goal language. To prevent collusion, namely a solution where a characteristic input graph encodes the goal schema using an elaborate scheme, the inference algorithm is required to be robust under extension: it must infer the goal schema even if the characteristic graph is accompanied by other potentially less informative graphs that satisfy the schema. Formally, *G extends G' consistently with schema H* iff there is G'' such that $G = G' \uplus G''$ and $G, G', G'' \in L(H)$.

▶ **Definition 3.** *A class of shape graphs $\mathcal{C}$ is* learnable *from a class of graphs $\mathcal{G}$ iff there is an inference algorithm* learner *such that*

Soundness *For every input graph $G \in \mathcal{G}$ the inference algorithm returns a schema $\mathsf{learner}(G) = H$ such that $H \in \mathcal{C}$ and $G \in L(H)$.*

Completeness *For every goal schema $H \in \mathcal{C}$ there exists a* characteristic graph *$G_H \in L(H)$ such that for any G that extends G_H consistently with H we have $\mathsf{learner}(G) \equiv H$.*

Furthermore, we say that $\mathcal{C}$ is learnable in polynomial time *if the inference algorithm works in polynomial time. We also say that $\mathcal{C}$ is* learnable in polynomial data *if there exists a polynomial that bounds the size of the characteristic graph as a function of the size of the goal schema.* ⌟

As an illustrative example of grammatical inference consider the problem of inferring basic intervals M_0 from a given sample of its members $\{i_1, \ldots, i_k\}$. Take the function $\mathsf{fit}_{\mathsf{M}_0}$ that returns the smallest basic interval that contains all the elements given on the input. Naturally, $\mathsf{fit}_{\mathsf{M}_0}$ is a sound inference algorithm since it returns an interval that contains all the elements given on the input. It is also a complete inference algorithm because we can construct characteristic samples for every element of M_0. For instance, for the goal interval $+ = [1; \infty]$ a characteristic sample is $\{1, 2\}$. Indeed, $\mathsf{fit}_{\mathsf{M}_0}(\{1, 2\}) = +$ but more importantly $\mathsf{fit}_{\mathsf{M}_0}(X) = +$ for any X subset of $[1; \infty]$ that includes $\{1, 2\}$.

Now, consider the full class of intervals I_0 and take the function $\mathsf{fit}_{\mathsf{I}_0}$ as a inference algorithm. It is naturally sound since $\mathsf{fit}_{\mathsf{I}_0}(X) = [\min(X); \max(X)]$ but it is not complete since it cannot infer an unbounded interval $[i_0; \infty]$ from a finite input set. In fact, I_0 is not learnable because one can show that there is no sound and complete inference algorithm for I_0 (from positive samples alone). Next, we present a general characterization of classes of languages that are not learnable. We will use it as a tool for identifying key challenges in schema inference.

3.2 Limit point

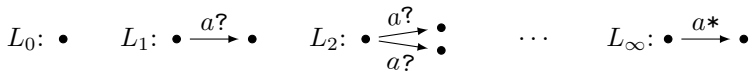
We recall that a class of languages $\mathcal{C}$ has the *limit point* property iff $\mathcal{C}$ contains an ascending chain of languages $L_1 \subsetneq L_2 \subsetneq \ldots$ whose limit point $L_\infty = \bigcup_i L_i$ also belongs to $\mathcal{C}$. It is a folklore result that the limit point property precludes learnability (from positive examples) [2]: essentially, no finite amount of examples from the goal language allows to distinguish between the limit point L_∞ and some language L_i in the ascending chain. We formally show that it also holds for the framework we adopt in this paper.

▶ **Proposition 4.** *No class of shape graphs with the limit point property is learnable from simple graphs.*

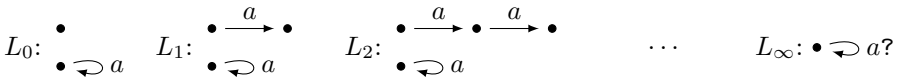
The limit point argument allows us to identify a number of limitations of inference of shape expressions schemas.

▶ **Lemma 5.** *For any $M \subseteq \{1, ?, *, +\}$ with at least two elements the class $\mathsf{ShEx}_0(M)$ has the limit point property.*

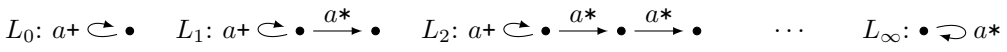
Proof sketch. It suffices to consider sets of 2 multiplicities and in Figure 5 we present three cases. The limit point arguments for all remaining cases are constructed very similarly to



(a) Limit point for $\mathsf{ShEx}(?, *)$.



(b) Limit point for $\mathsf{ShEx}(1, ?)$.



(c) Limit point for $\mathsf{ShEx}(*, +)$.

Figure 5 Limit points for various subclasses of ShEx_0.

$\mathsf{ShEx}(?, *)$ in Figure 5a (details in appendix). ◀

Consequently, we obtain the following corollary.

▶ **Corollary 6.** ShEx_0 *is not learnable from simple graphs.*

A closer look at the limit point arguments offers an insight into the reasons why inferring ShEx_0 is not feasible. Firstly, we observe in Figure 5a that one of the challenges in distinguishing between L_i and L_∞ is in deciding based on finite input example when to use unbound arity $*$ as opposed to $?$ (similar argument holds for $+$ and $?$, etc.). Secondly, in Figures 5b and 5c we observe an analogous difficulty in deciding between other pairs of arities, but more importantly, we also observe a different difficulty: whether to choose a simple recursive type (L_∞) or allow some arbitrary non-recursive unfolding of it (L_i). To summarize, the constructions in Figure 5 point to two key challenges of inferring shape schemas for graph databases: 1) distinguishing between bounded and unbounded arities, and 2) introducing recursive types especially from a possibly acyclic input example.

3.3 Graph aggregate

The inference algorithms that we propose in this paper construct schemas with an operation that generalizes the standard graph quotient. Recall that the graph quotient operation essentially reduces to a single node every set of nodes that are equivalent w.r.t. a given equivalence relation. In the context of schema inference, we are interested in grouping together nodes that we deem to have the same type but we point out that the underlying relation "*n and m are deemed to have the same type*" needs not be transitive in general. In fact, we do not even assume that this relation is reflexive because we may wish to intentionally ignore nodes of the input graph that we do not find informative enough.

▶ **Definition 7.** *A* node grouping *of a simple graph $G = (N_G, E_G, \mathsf{arity}_G)$ is a collection $P \subseteq 2^{N_G}$ of sets of nodes of G such that any $N \in P$ with $N \cap \mathcal{L} \neq \emptyset$ is a singleton. Given a proper set of intervals M, the* aggregate *of G by P w.r.t. M, in symbols $G \div_M P$, is the shape graph $H = (N_H, E_H, \mathsf{arity}_H)$, where*

$$N_H = P,$$
$$E_H = \{(N, a, N') \in P \times \Sigma \times P \mid \exists (n, a, n') \in E_G,\ n \in N \wedge n' \in N'\},$$
$$\mathsf{arity}_H(N, a, N') = \mathsf{fit}_M\left(\left\{\bigoplus\{\mathsf{arity}_G(n, a, n') \mid n' \in N',\ (n, a, n') \in E_G\} \mid n \in N\right\}\right).$$

The requirement $N \cap \mathcal{L} \neq \emptyset$ follows from the dichotomy of node kinds and their types. To ensure that the aggregate is technically a (shape) graph, we assume that any subset of $\mathcal{N}$ is also a node identifier and for any $\ell \in \mathcal{L}$ we interpret $\{\ell\}$ as ℓ. In the sequel, when the set of intervals M is known from the context, we omit it in the aggregate and simply write $G \div P$.

In essence, for a given input graph G, our inference algorithms explore the lattice $(\mathcal{P}_G, \sqsubseteq)$ of all node groupings $\mathcal{P}_G$ of G from $P_\bot = \{\{n\} \mid n \in N_G\}$ to $P_\top = \{N_G \setminus \mathcal{L}\} \cup \{\{\ell\} \mid \ell \in N_G \cap \mathcal{L}\}$. The structure of this lattice is consistent with the containment relation: $P_1 \sqsubseteq P_2$ iff $G \div P_1 \subseteq G \div P_2$. This connection to containment is essential for guiding inference algorithms with characteristic graphs. When the goal schema is $G \div P$ for a node grouping P then for every relevant $P' \sqsubseteq P$ the characteristic graph needs to contain an example showing $G \div P' \not\supseteq G \div P$.

▶ **Example 8.** Consider G_0 and H_0 in Figure 2b, and take the node grouping that corresponds to the typing of G_0 w.r.t. H_0: $P = \{\{\mathtt{b}_1, \mathtt{b}_2, \mathtt{b}_3, \mathtt{b}_4\}, \{\mathtt{u}_1, \mathtt{u}_2, \mathtt{e}_1\}, \{\mathtt{e}_1\}, \{\mathsf{str}\}\}$. The aggregate $G_0 \div_{\mathsf{M}_0} P$ is in fact (equivalent to) H_0. ⌟

The above example shows that G_0 carries sufficient information that with the right partition allows to infer the goal schema H_0.

The task is to identify a method that allows to group nodes of the input graph that are deemed to have the same type and at the same time can introduce unbounded arities and recursion in type definitions. A natural attempt at grouping nodes of a graph would be to use bisimulation. But as we illustrate in Figure 4 this fails to introduce unbounded arities and recursive types. In fact, it has been studied in the literature [27] and found to fail to generalize appropriately the input graph.

4 Singular shape graphs

In this section we address the two key challenges of inference by grouping nodes with the help of graph simulation relation. This yields an inference algorithm for a subclass of singular shape graphs. The inference algorithm allows to infer unbounded arities and recursive types even from acyclic input graph. The price is the ability to distinguish types from their subtypes.

Shape graphs assign types to nodes using an embedding, which is an adaptation of the standard notion of graph simulation. Consequently, we explore an approach of grouping together nodes that are comparable with the simulation relation. To that effect, for a given simple graph G we construct the graph $G*$ by changing the arity of every edge to $*$, and then, any autoembedding of $G*$ is a simulation relation on G and vice versa. Hence, we use the maximal autoembedding $\ll_{G*}$ to assess the relative amount of schema information that a node supplies: if $m \ll_{G*} n$, we consider m to provide no more schema information than n does. We introduce useful short-hands. We say that n *sim-dominates* m iff $m \ll_{G*} n$ and $m \neq n$. Also, n *properly sim-dominates* m is n *sim-dominates* m and $n \not\ll_{G*} m$. We say that n and m are *sim-equivalent* if $m \ll_{G*} n$ and $n \ll_{G*} m$. Finally, a node is *sim-maximal* if it is sim-dominated only by sim-equivalent nodes.

Using simulation to group nodes has an important drawback: the inability to distinguish types from their subtypes. Indeed, if $t_1 \subseteq t_2$, then any node that has type t_1 is simulated by nodes of type t_2 that characterize the type t_2. We return to this problem in Section 6 where we show how to handle inference of restricted cases of subtypes. We point out that permitting arbitrary subtypes allows to construct ascending chains, which can be then easily used in limit point arguments, as evidenced for instance in Figure 5b. This allows us to conclude that inferring subtypes is inherently problematic. Furthermore, we show that even forbidding subtypes does not suffice to guarantee learnability.

▶ **Proposition 9.** *Let* $\mathsf{FlatShEx}_0$ *be the subclass of shape graphs where no two different types are comparable by the containment relation. For any subset* $M \subseteq \mathsf{M}_0$ *with at least two elements* $\mathsf{FlatShEx}_0(M)$ *is not learnable.*

Consequently, stronger restrictions are needed and we propose the class of singular schemas whose types can all be distinguished with the graph simulation relation.

▶ **Definition 10.** *A type* t *of a shape graph* H *is* singular *iff* t *is not sim-dominated by any type in* H*. A shape graph is* singular *iff it has only singular types and by* $\mathsf{SinShEx}_0$ *we denote the set of all singular shape graphs.*

Intuitively, singularity ensures that for any two types t_1 and t_2, there exist nodes of type t_1 that can be distinguished from all nodes of type t_2 with the help of simulation. In the following example we show that simulation needs to be used carefully when identifying groups of nodes of the same type.

▶ **Example 11.** To keep the example simple, we limit the intervals used in shape graphs to $\{1, ?, *\}$. In figures we use a dotted green arrow $n \dashleftarrow m$ to indicate that n sim-dominates m.

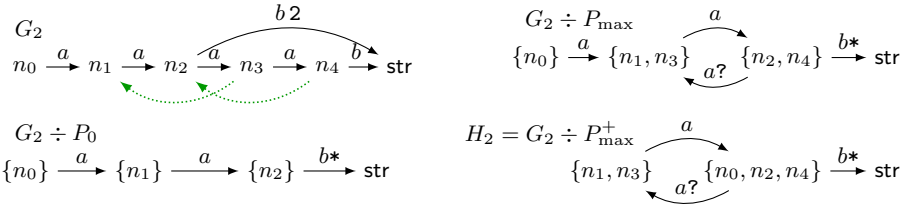


Figure 6 Example of inference using graph simulation. $G_2 \div P_0$ does not recognize G_2.

Take the graph G_2 in Figure 6, identify its sim-maximal nodes $Max = \{n_0, n_1, n_2, \mathsf{str}\}$, and consider the node grouping $P_0 = \{\{n_0\}, \{n_1\}, \{n_2\}, \{\mathsf{str}\}\}$. This grouping is too sparse: the aggregate $G_2 \div P_0$ does not recognize G_2. Consider then the grouping where every sim-maximal node is accompanied by all sim-dominated nodes: $P_{\max} = \{\{n_0\}, \{n_1, n_3\}, \{n_2, n_4\}, \{\mathsf{str}\}\}$. While $G_2 \div P_{\max}$ recognizes G_2, it is not singular because $\{n_0\} \ll_{G_2 \div P_{\max}*} \{n_2, n_4\}$. Although originally n_0 was sim-maximal, afterwards it became subsumed by the group $\{n_2, n_4\}$. It is then absorbed by $\{n_2, n_4\}$ and we get the node grouping $P^+_{\max} = \{\{n_1, n_3\}, \{n_0, n_2, n_4\}, \{\mathsf{str}\}\}$, which yields the singular shape graph H_2 that recognizes G_2.

The node grouping $P_{\max}$ is not a good starting point for an inference algorithm because it might be overly eager as we illustrate next. Consider the graph G_3 in Figure 7. The node m_2 is sim-dominated by m_1 and m_3. The grouping $P_{\max} = \{\{m_0\}, \{m_1, m_2\}, \{m_2, m_3, m_4\}, \{\mathsf{str}\}\}$ forces $*$ on the edge from $\{m_0\}$ to $\{m_1, m_2\}$. It is, however, not necessary to add m_2 to any group: $P_1 = \{\{m_0\}, \{m_1\}, \{m_3, m_4\}, \{\mathsf{str}\}\}$ already yields H_3 that is singular, recognizes G_3, and is a tighter fit than $G_3 \div P_{\max}$. ⌟

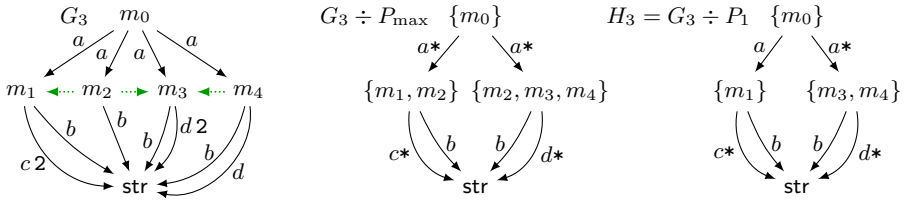


Figure 7 Example of inference using graph simulation. $G_3 \div P_{\max}$ is overgeneralized.

For inferring singular shape graphs we propose the algorithm sim-learner. It is parameterized by a proper set of intervals M which restricts the intervals used in the output singular shape graph.

Algorithm 1 Inference algorithm for $\mathsf{SinShEx}_0(M)$.

algorithm $\mathsf{sim\text{-}learner}_M(G)$
input: $G = (N_G, E_G)$ a simple graph
param: M a proper set of intervals
output: inferred shape graph in $\mathsf{SinShEx}_0(M)$

begin

1: $Max := \{n \in N_G \mid \forall m \in N_G.\ n \ll_{G*} m \Rightarrow m \ll_{G*} n\}$
2: $P := \varnothing$
3: **for** $n \in Max$ **do**
4: $\quad P[n] := \{m \in Max \mid m \ll_{G*} n\}$
5: **while** $G \not\preccurlyeq G \div_M P$ **do**
6: $\quad$ Choose $n \in Max$ and $m \in N_G \setminus P[n]$ based on priority rules Φ, such that $m \ll_{G*} n$
7: $\quad$ add m to $P[n]$
8: **while** $\exists S, T \in P.\ S \neq T \wedge S \ll_{(G \div_M P)*} T$ **do**
9: $\quad$ merge S and T in P
10: **return** $G \div_M P$

end

The algorithm constructs a node grouping P of the input graph G and it constructs only groupings where every sim-maximal node n belongs to precisely one node group, which can be accessed with a lookup operation $P[n]$. The algorithm begins by identifying all sim-maximal nodes (line 1) and grouping together all sim-equivalent nodes (lines 2-4). Next, as long as the aggregate $G \div_M P$ does not recognize the input graph, the algorithm iteratively identifies a node m that is absorbed by a node group $P[n]$ such that $m \notin P[n]$ (lines 5-7). Among all eligible pairs of m and $P[n]$ we choose one according to the following order Φ:

1. **siblings first** any m that has a sibling $n \in Max$ such that $P[n]$ contains all siblings of m that sim-dominate it.
2. **single choice** any m and $n \in Max$ such that all nodes that sim-dominate m are in $P[n]$.
3. **all else** any $n \in Max$ and $m \in N_G$ such that $m \in N_G \setminus P[n]$ and n sim-dominates m.

Finally, the algorithm merges any node groups that prevent the shape graph $G \div_M P$ from being singular (lines 8-10). We point out that the preference order Φ may indicate multiple equally preferred pairs and the algorithm may choose an arbitrary one: regardless of the choice the output always recognizes the input graph and when the input contains a characteristic graph, the algorithm will always converge on the goal schema. Φ prevents the algorithm sim-learner from performing unnecessary generalization: when executed on the

graph G_3 in Figure 7 it returns the schema H_3. Furthermore, when run on G_2 in Figure 6 the algorithm returns the schema H_2, which illustrates that it is capable of inferring recursive types even from acyclic input graphs.

In general the algorithm is sound for $\mathsf{SinShEx_0}$, but here we only claim that $\mathsf{sim\text{-}learner}_{1,*}$ is sound and complete for the shape graphs using 1 and $*$.

▶ **Theorem 12.** $\mathsf{SinShEx_0}(1, *)$ *is learnable in polynomial time and data from simple graphs.*

In Section 6.2 we take advantage of Φ to show completeness for a larger class but it remains an open question if the full class of singular shape graphs is learnable in polynomial time and data (but the existing evidence suggests it is unlikely [49]). The full class is however learnable with an algorithm that exhaustively explores the exponential space of possible singular shape graphs and uses an exponentially-large characteristic graph to help navigate this space.

▶ **Theorem 13.** $\mathsf{SinShEx_0}$ *is learnable from simple graphs.*

5 Deterministic shape graphs

In this section we investigate the potential ramifications of attempting to improve the process of identifying the set of types by exploiting the context in which nodes are used in the graph. More precisely, two nodes n and n' are used in the same *context* of node m and label a if there are two edges (m, a, n) and (m, a, n'), and in that case the nodes n and n' should have the same type. Such premise leads us to the class of deterministic shape expressions schemas that allow at most one type to be used in a given context.

▶ **Definition 14.** *A type t of a shape graph H is* deterministic *iff for every label $a \in \Sigma$ the type t has at most one outgoing edge with label a. A* deterministic *shape graph has only deterministic types and by* $\mathsf{DetShEx_0}$ *we denote the set of all deterministic shape graphs.*

Interestingly, the powerset method for determinization of finite automata can be adopted for inference of $\mathsf{DetShEx_0}$. We illustrate this algorithm in Figure 8 but refrain from formally introducing it because we do not wish to promote it in this paper.

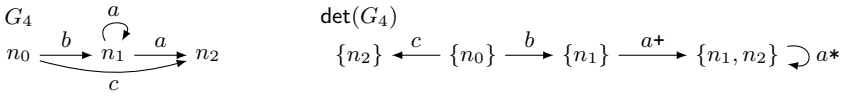


Figure 8 Determinization of a graph.

Later in this section we provide its alternative semantic characterization that will allow us to identify the pitfalls of an indiscriminate use of context information for shape graph inference.

Some caveats follow. Firstly, not every simple graph can be recognized by a deterministic shape graph. For instance, the graph G_1 in Figure 2a satisfies no deterministic schema because G_1 has a node with two outgoing b-edges, one of which leads to a literal node and the other to a non-literal node. Consequently, we limit the inputs to *data-free graphs*, simple graphs with no literal nodes. Secondly, the limit point in Figure 5b shows that $\mathsf{DetShEx_0}$ is not learnable, and thus, we recall from [49] a subclass $\mathsf{DetShEx_0^-}$ that we show learnable with the determinization algorithm. Formally, a *reference* to a type $t \in N_H$ is any edge $e \in E_H$

that leads to t i.e. $\mathsf{target}_H(e) = t$. A reference e is $*$-*closed* if $\mathsf{arity}_H(e) = *$ or all references to $\mathsf{source}_H(e)$ are $*$-closed. $\mathsf{DetShEx}_0^-$ is the class of deterministic shape graphs that do not use + and any type using ? is referenced at least once and all references to it are $*$-closed.

▶ **Theorem 15.** $\mathsf{DetShEx}_0$ *is not learnable from data-free graphs.* $\mathsf{DetShEx}_0^-$ *is learnable from data-free graphs in polynomial data.*

It can be proved (see appendix) that the determinization algorithm outputs the containment-minimal shape graph in $\mathsf{DetShEx}_0^-$ that satisfies the input data-free graph G. This makes the determinization algorithm an embodiment of the *minicon* principle which postulates for constructing the minimal consistent schema. This principle has a negative ramifications.

▶ **Proposition 16.** *For any n there exists a graph G_n of size quadratic in n such that the containment-minimal shape graph in* $\mathsf{DetShEx}_0^-$ *that validates G is of size exponential in n.*

We conclude that an indiscriminate use of the context information has negative impact on inference of shape graphs: the obtained schemas risk being too verbose and overfitted. Also, nontrivial classes of shape graphs are too rich to benefit from blindly following the minicon principle.

6 Hybrid approach

In this section we pursue a hybrid approach that extends the simulation-based approach with a limited amount of context information to enable inference of shape graphs with subtypes. We then identify the subclass that our hybrid algorithm is able to infer among shape graphs using arbitrary basic intervals.

6.1 Context-singular shape graphs

We extend the simulation-based approach with context information limited to the labels of incoming edges. This allows to infer subtypes as long as they are used in different contexts. For instance, this extension can infer the schema in Figure 1, where Employee $\subseteq$ User but Employee is used in the context of verifiedBy and User is used only in the context submittedBy.

Here, the contexts in which a type t is used in a shape graph H are described by the labels of incoming edges: $\mathit{ctx\text{-}lab}_H(t) = \{a \in \Sigma \mid \exists s \in N_H.\ (s, a, t) \in E_H\}$. Singularity of shape graph requires that no type is sim-dominated by another type. We relax this by allowing a sim-dominated type as long as it is used in at least one context where none of its sim-dominating types are.

▶ **Definition 17.** *Given a shape graph H, a type t of H is* context-singular *if and only if* $\mathit{ctx\text{-}lab}_H(t) \not\subseteq \bigcup\{\mathit{ctx\text{-}lab}_H(t') \mid t' \in N_H,\ t \ll_{H*} t',\ t \neq t'\}$. *$H$ is* context-singular *if it has only context-singular types.* $\mathsf{CtxSinShEx}_0$ *is the set of all context-singular shape graphs.*

To adapt sim-learner to inference of context-singular shape graphs we introduce an uncoupling operation that for a given graph creates a distinct clone of a node for every of its incoming edges, so that all incoming edges of a node have the same label. Formally, the

uncoupling of a graph G is $\bar{G} = (N_{\bar{G}}, E_{\bar{G}}, \mathsf{arity}_{\bar{G}})$, where

$$N_{\bar{G}} = \bigcup\{uncouple(n) \mid n \in N_G\},$$

$$uncouple(n) = \begin{cases} \{n\} & \text{if } n \in \mathcal{L}, \\ \{(\bot, n)\} & \text{if } ctx\text{-}lab_G(n) = \emptyset, \\ \{(a, n) \mid a \in ctx\text{-}lab_G(n)\} & \text{otherwise,} \end{cases}$$

$$E_{\bar{G}} = \{(\eta, a, trg(a, m)) \mid (n, a, m) \in E_G,\ \eta \in uncouple(n)\}$$

$$trg(a, m) = \begin{cases} m & \text{if } m \in \mathcal{L}, \\ (a, m) & \text{otherwise,} \end{cases}$$

$$\mathsf{arity}_{\bar{G}}(\eta, a, trg(a, m)) = \mathsf{arity}_G(n, a, m) \quad \text{for } (n, a, m) \in E_G \text{ and } \eta \in uncouple(n).$$

When each node has incoming edges with the same label, using the context information is simpler. For instance, a type t of an uncoupled shape graph H is context-singular iff there is no type t' such that $t \ll_{H*} t'$ and $ctx\text{-}lab_H(t) \subseteq ctx\text{-}lab_H(t')$. Consequently, for uncoupled graphs we define a contextualized variant of the autoembedding relation: $n \ll_K^{ctx} m$ iff $n \ll_K m$ and $ctx\text{-}lab_K(n) \subseteq ctx\text{-}lab_K(m)$. We adapt terminology and say that n *ctx-dominates* m iff $m \ll_K^{ctx} n$ and $n \neq m$. Also, n is *ctx-maximal* if any node that ctx-dominates n is also ctx-dominated by n.

Now, the inference algorithm hybrid-learner is a modification of sim-learner where we replace every occurrence of G by $\bar{G}$ and every occurrence of $\ll$ by $\ll^{ctx}$. We illustrate hybrid-learner on an example.

▶ **Example 18.** We run hybrid-learner in Figure 9 on the graph G_0 introduced in Figure 2. In the uncoupling of G_0 we indicate with $m \dashrightarrow n$ that $m \ll^{ctx}_{\overline{G_0}*} n$.

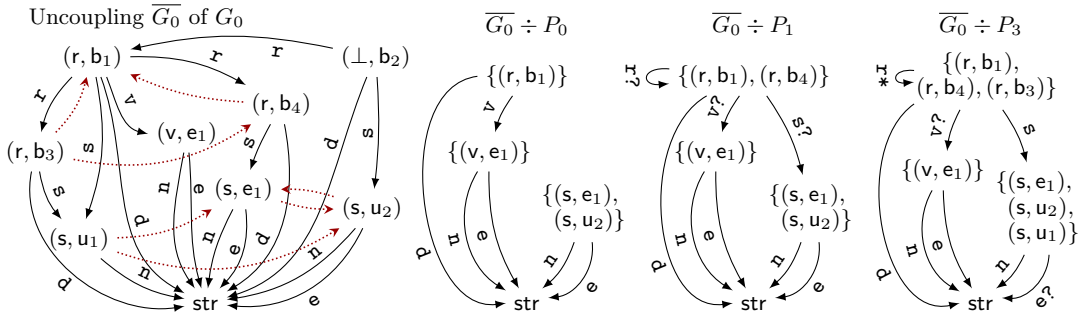


Figure 9 Execution of hybrid-learner on G_0. $m \dashrightarrow n$ means that n sim-dominates m.

In this example we use the complete set of basic intervals $\mathsf{M}_0 = \{1, ?, +, *\}$. The algorithm begins with the node grouping $P_0 = \{\{(\mathsf{r}, \mathsf{b}_1)\}, \{(\mathsf{v}, \mathsf{e}_1)\}, \{(\mathsf{s}, \mathsf{e}_1), (\mathsf{s}, \mathsf{u}_2)\}, \{\mathsf{str}\}\}$ obtained from the ctx-maximal nodes in $\overline{G_0}$. Because $\overline{G_0} \div P_0$ does not recognize $\overline{G_0}$, the algorithm uses Φ to identify nodes that can be absorbed: $(\mathsf{r}, \mathsf{b}_4)$ can be absorbed in $P_0[(\mathsf{r}, \mathsf{b}_1)]$ only. After this operation the node grouping P_1 still does not yield a shape graph that recognizes $\overline{G_0}$. In the next step, $(\mathsf{r}, \mathsf{b}_3)$ being ctx-dominated by its sibling $(\mathsf{r}, \mathsf{b}_4)$ is absorbed in $P_1[(\mathsf{r}, \mathsf{b}_1)]$. The resulting node grouping P_2 again fails to recognize $\overline{G_0}$. Finally, the node $(\mathsf{s}, \mathsf{u}_1)$ is absorbed in $P_2[(\mathsf{s}, \mathsf{e}_1)]$ and the resulting node grouping P_3 yields $\overline{G_0} \div P_3 \equiv H_0$ which recognizes $\overline{G_0}$. ⌟

The algorithm hybrid-learner is a sound inference algorithm for $\mathsf{CtxSinShEx}_0$ and analogously to sim-learner it is also complete for $\mathsf{CtxSinShEx}_0(1, *)$. The full class of context-singular shape graphs is learnable with an exhaustive exploration of the exponential search space with exponentially large characteristic graphs.

▶ **Theorem 19.** $\mathsf{CtxSinShEx}_0(1, *)$ *is learnable in polynomial time and data from simple graphs.* $\mathsf{CtxSinShEx}_0$ *is learnable from simple graphs.*

6.2 Absorption-protected shape graphs

In this section we characterize the subclass of $\mathsf{CtxSinShEx}_0$, using all basic intervals $\mathsf{M}_0 = \{1, ?, +, *\}$, for which hybrid-learner is sound and complete. To do that we first identify the information that needs to be contained in the input graph for hybrid-learner to infer the right arity on the edge of a the goal shape graph.

Let us fix a goal shape graph H and an edge $(t, a, s) \in E_H$. For hybrid-learner to output H, it must be able to construct from a characteristic graph G two groups of nodes P_t and P_s that characterize the interval $\mathsf{arity}_H(t, a, s)$. In particular, they must contain sufficient information to indicate whether the interval is unbounded ($+$ and $*$) and whether it is nullable ($?$ and $*$).

Characterizing an unbounded interval is relatively simple. It suffices to introduce in G a ctx-maximal node n_t of type t that has at least two outgoing a-edges to nodes n_s and m_s of type s, one of which, say n_s should also be ctx-maximal. Naturally, the nodes n_t and n_s will be put by hybrid-learner in the groups P_t and P_s respectively. Because of the first rule of the preference order Φ, the node m_s will also be added to P_m.

Characterizing a nullable interval is more challenging since it requires making sure that P_t and P_s absorb information that is more volatile. Namely, we need to make sure that G has a ctx-maximal node n_t and a node m_t of type t without any a-edge to a node of type s. Since n_t is ctx-maximal, it will be absorbed in P_t but since m_t is not a ctx-maximal node, G must protect it from being absorbed by any group other than P_t. The node m_t can be absorbed into a group $P_{s'}$, representing a type s', if s' dominates a version η of the type t from which we remove the edge (t, a, s). Naturally, m_t is protected if no such node exists, and then the second rule of Φ ensures that m_t ends up in P_t. Otherwise, the type t allows protection from an absorption if t has an incoming edge with unbounded arity and none of the siblings of t dominate η. Then m_t also ends up in P_t because of the first rule of Φ for same reasons as explained for unbounded intervals.

To characterize the classes of schemas that allow protection from absorption we need to introduce a construction of the type η. Formally, let H be a shape graph H, $\eta \in \mathcal{N} \setminus N_H$ a fresh node identifier, and $e \in E_H$ an edge from t to s with unbounded arity. The *absorption-protection test graph* $\mathsf{APT}^*(H, e, \eta)$ is a shape graph H' defined as:

$$N_{H'} = N_H \cup \{\eta\}, \quad E_{H'} = E_H \cup \{(\eta, a, s') | (t, a, s') \in \mathsf{out}_H(t) \setminus \{e\}\}, \quad \mathsf{arity}_{H'}(f) = * \text{ for } f \in E_{H'}.$$

Essentially, we add the type η, for every outgoing edge e' from t other than e, we add an outgoing edge with the same label as e' from η to the same node as the endpoint of e', and we set arities of all edges to $*$. Next, we formally characterize the class of shape graphs that hybrid-learner is capable of identifying (we use sim-domination because η in $\mathsf{APT}^*(H, e, \eta)$ does not have any incoming edges).

▶ **Definition 20.** *A type t of a shape graph H is* absorption-protected *iff for every edge $e \in \mathsf{out}_H(t)$ with nullable arity one of the following conditions holds:*

1. *t is the only node that sim-dominates η in $\mathsf{APT}^*(H, e, \eta)$ or,*
2. *t has an incoming edge with unbounded arity and t is the only node among its siblings that sim-dominates η in $\mathsf{APT}^*(H, e, \eta)$.*

An absorption-protected *shape graph is a context-singular shape graph that has only absorption-protected types. By* $\mathsf{AbsShEx}_0$ *we denote the set of all absorption-protected shape graphs.*

We illustrate absorption protection in the following example.

▶ **Example 21.** In Figure 10, H_1 is not absorption-protected because of the edge $e = (t_1, a?, \mathsf{str})$. The copy η of t_1 without the edge e is sim-dominated by the type t_2 of H_1. H_2 is absorption-protected. Even if for the type s_1 we create a copy η without the edge $(s_1, a?, \mathsf{str})$ and find it is sim-dominated by s_2, the type s_1 has an incoming unbounded r-edge and the type s_2 is not a sibling of s_1.

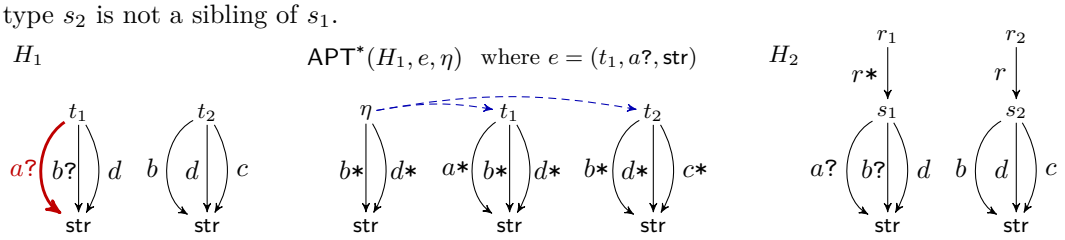


Figure 10 Two shape graphs, H_1 is not absorption-protected because t_1 is not, but H_2 is.

We point out that the shape graph in Figure 1 is absorption protected, which explains why in Example 18 the algorithm correctly infers the schema H_0. We finish by formally stating the learnability result.

▶ **Theorem 22.** $\mathsf{AbsShEx}_0$ *is learnable in polynomial time and data from simple graphs.*

7 Schemas of existing graph databases

In this section we present an analysis (Table 1) of a number of graph databases in order to find out to what extent their schemas can be captured with shape graphs and the subclasses that we propose, and consequently, to assess the usefulness of the proposed inference algorithms. We have analyzed: the Berlin SPARQL Benchmark [10], the Tpc-h benchmark converted to RDF with Direct Mapping [51, 54], the Nobel Prize database [4], an RDF serialization of the Dblp database (retrieved from `https://www.rdfhdt.org/datasets/` on 14.01.2021), the Springer Nature knowledge graph SciGraph [32], the Covid19 knowledge graph [43], and SHACL constraints in the version 4 of Yago knowledge base [50]. For each database we have defined appropriate schemas using ShEx, which is a fully fledged schema language and properly contains ShEx_0 (cf. [47]). Complete schemas and more details can be found in the appendix.

Table 1 Analysis of schemas for a number of graph databases.

	Base types						Union	Clauses		
	total	Det	σ-ind	Sin	CtxSin	Abs	types	total	union	disj.
Dblp	11	5	6	8	10	7	2 (3)	135	2 (9)	1
Berlin	8	8	5	5	8	7	0 (0)	47	0 (0)	0
Tpc-h (dm)	8	8	8	8	8	8	0 (0)	67	0 (0)	0
Nobel	11	8	7	7	10	8	1 (0)	66	2 (0)	1
SciGraph	18	10	16	17	18	16	1 (2)	162	1 (7)	1
Covid19	22	11	11	19	19	19	2 (0)	295	3 (0)	0
Yago	49	14	28	28	30	21	2 (13)	1174	4 (490)	0

We have found that the ShEx schemas for the above databases require the use of *disjunction* and *union types*, features that are absent in ShEx_0 [47, 55]. We illustrate them on the following example inspired by DBLP.

$\mathsf{Book} \to (\mathit{creator} : \mathsf{Person+} \mid \mathit{editor} : \mathsf{Person+}),\ \mathit{title} : \mathsf{Literal},\ \mathit{ref} : (\mathsf{Article} \cup \mathsf{Book})*.$

It states that a book has either creators (authors) or editors but never both, has a single title, and references an arbitrary number of articles and books. We say that this base type definition has 3 clauses, one uses disjunction, and one uses a union type. Union types are *trivial* when used only under $*$ since $a : (t_1 \cup t_2)*$ is equivalent to $a : t_1*,\ a : t_2*$. For every schema we have measured the quantities of the following objects: nontrivial (and trivial) union types, clauses, clauses with disjunction, clauses with nontrivial (and trivial) use of a union type.

To help assessing the usefulness of the proposed inference algorithms, for every ShEx schema we have constructed its straightforward ShEx_0-approximation as illustrated on the example for the above type definition of Book

$$\mathsf{Book} \to \mathit{creator} : \mathsf{Person}*,\ \mathit{editor} : \mathsf{Person}*,\ \mathit{title} : \mathsf{Literal},\ \mathit{ref} : \mathsf{Article}*,\ \mathit{ref} : \mathsf{Book}*.$$

Then in each ShEx_0 we have measured the number of all (base) types, those that are singular (Sin), context-singular (CtxSin), and absorption-protected (Abs). Additionally, we have counted the number of types that are signature-independent (σ-ind): types whose set of labels of outgoing edges is incomparable with others.

We first analyze our findings for the graph databases and knowledge graphs (first 6 rows). We observe that the numbers of nontrivial union types and disjunctions are relatively small. Also, deterministic types are relatively low in some databases because of use of union types: even trivial use of union types in a type definition precludes determinism. We point to the differences in signature-independent and singular types, on average 69% vs. 79% resp., which indicates that using simulation to identify types has advantages over an approach that reduces a type to the labels of its outgoing types. While the numbers of singular types are relatively large, 79% on average, the numbers of context-singular types, 94% on average, are even more impressive, and the significant improvement validates our approach of incorporating the context information in the inference process.

For the knowledge base YAGO the results indicate that our algorithms would offer little improvement over simpler approaches (σ-ind) that attempt to identify types by using the labels of their outgoing edges. We attribute this to the use of rich hierarchies of classes which introduce many subtype relationships between types. We point out, however, that those class hierarchies are defined with ontologies that specify rules that allow to type the graph. Our methods do not take advantage of this typing information, and therefore, are not geared towards inference for knowledge bases. Alternative tools for inference from typed graphs should be used (see our preliminary results on the topic in [30]).

We conclude that inference algorithms for the proposed classes $\mathsf{CtxSinShEx}_0$ and $\mathsf{AbsShEx}_0$ are an attractive choice for the first draft of the schema for (untyped) graph databases and knowledge graphs to be later on improved by a data specialist, especially when the type hierarchy is relatively flat.

8 Related work

Several approaches have been investigated in the late 90's to summarize graph data, defining schema models, and algorithms to infer schemas from graph data, then viewed as semi-structured data [15, 24, 41, 40]. A large part of the literature focuses on graph summarization [14], where the purpose is to obtain a concise representation of an input graph that can be used to formulate queries against the data. Our approach takes a different perspective of the same problem, where the input data is considered as evidence from which we must infer an unknown target schema. Using the grammatical inference framework enables us to offer strong guarantees, and lay the ground for a principled fundamental approach. To

the best of our knowledge this is a first, and this complements the traditional approach in the literature that consists in evaluating algorithms empirically on real-life graph data to measure the tradeoff between compactness and the accuracy of the description.

A *Dataguide* [29] of a graph represents as a DFA all the paths that can be navigated from a root node in the graph. Dataguides are thus similar in spirit to our determinization procedure, but do not represent the arity of paths and assume that specific entry points have been specified; the root nodes. This approach, like determinization, can identify subtypes, and cycles (provided the cycles are presented in the input data). Several approaches have been proposed to infer schemas for tree-structured data [25, 39, 21, 8, 9] or JSON data [5, 6]. The schemas inferred include DTDs, tree automata, and XML Schema, and thus are expressive enough to represent the arity of edges. However, these approaches focus on tree-structure and do not consider cycles (as expressed via mechanisms such as ID/REF). The tree structure means that the schema can be inferred recursively, and thus the task is essentially about inferring efficiently a regular expression representing for the children of each tree node.

More recently, the inference of schemas for graphs has focused on RDF and property graphs [34]. Several applications [23, 56, 33, 52] rely on graph (bi)simulation to build a quotient graph w.r.t. an equivalence relation. The quotient graph merges all nodes that belong to the same equivalence class. Those proposals generally do not assign arities to the edges with a few exceptions such as [53]. Variants [16, 44, 52] consider both forward and backward bisimilarity, which means that types can be defined based on both the incoming and outgoing edges. Full bisimulation is rather rare in heterogenous graphs and even limiting it to the 1-hop neighbourhood does not help [26, 20, 18, 27], so the authors introduce weak and strong equivalence relations that relax rather aggressively the bisimulation relation through transitive co-occurrence of edge labels, to achieve more compact graph summaries. Type hierarchies essentially collapse in summaries built on weak or strong equivalence whenever they would collapse for hybrid-learner. Actually, weak equivalence systematically collapses hierarchies (except for nodes without outgoing edges). Even these summaries may become large for big, encyclopedic graphs, such as DBPedia or YAGO. One solution they suggest is to use methods based on mining frequent patterns [19]. An alternative approach to such problematic graphs is to use semi automatic methods where the construction of schema is interactively guided by domain experts [12].

Other ideas include clustering nodes according to thresholds on the number of shared edge labels [53]. In [3] integer linear programming is used to optimize the tradeoff between the number of partitions and how well each partition fits to the schema, however, the fitness is defined using outbound neighborhood, and not the context. Finally, some approaches complement the structural information with types that may be already present in graphs [16, 27] or, in the case of property graphs, by inferring the typing from node labels [37].

Among results on grammatical inference, [31] shows that a family of well-behaved inference algorithms can be interpreted as constructing a least upper bound for the input in the lattice of languages from the class, where the order is defined as the containment of languages. Our inference algorithm follows a similar approach. It does not compute a least upper bound among the whole class of graph schemas, but rather inside the lattice of the graph aggregates of the input graph (see Section 3.3).

9 Conclusions and future work

We have presented a principled approach to the problem of inference of shape graphs, which are a subclass of shape expression schemas for graph databases. We have employed the framework of grammatical inference to identify key challenges: identifying arities on edges and identifying recursive types. To confront these challenges we have explored an approach of inferring shape graphs using simulation relation on the nodes of the input graph. This

approach addresses well the identified challenges at the price of identifying subtypes, which we argue to be a fundamental limitation. To overcome it we extend the simulation based approach with a modest amount of context information. The resulting inference algorithm allows to infer a rich class of schemas, which we argue to be of practical importance with an empirical study of real-life schemas of graph databases.

Future research directions include exploring techniques for inference of disjunctions [11] and adapting the proposed methods to the emerging standard for Property Graph Schemas. We also plan to pursue the question of inference of shape graphs from typed graphs, our preliminary results are very promising [30].

References

1 S. Abiteboul, M. Arenas, P. Barceló, M. Bienvenu, D. Calvanese, C. David, R. Hull, E. Hüllermeier, B. Kimelfeld, L. Libkin, W. Martens, T. Milo, F. Murlak, F. Neven, M. Ortiz, T. Schwentick, J. Stoyanovich, J. Su, D. Suciu, V. Vianu, and K. Yi. Research directions for principles of data management (Dagstuhl Perspectives Workshop 16151). *Dagstuhl Manifestos*, 7(1):1–29, 2018.

2 D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

3 M. Arenas, G. I. Diaz, A. Fokoue, A. Kementsietsidis, and K. Srinivas. A principled approach to bridging the gap between graph data and their schemas. *Proceedings of the VLDB Endowment*, 7(8):601–612, 2014.

4 R. Asif and M. A. Qadir. Enhancing the Nobel Prize schema. In *International Conference on Communication, Computing and Digital Systems (C-CODE)*, pages 193–198, 2017.

5 M. A. Baazizi, H. Ben Lahmar, D. Colazzo, G. Ghelli, and C. Sartiani. Schema inference for massive JSON datasets. In *International Conference on Extending Database Technology (EDBT)*, 2017.

6 M. A. Baazizi, D. Colazzo, G. Ghelli, and C. Sartiani. Parametric schema inference for massive JSON datasets. *The VLDB Journal*, 28(4):497–521, 2019.

7 G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Transactions on the Web*, 4(4):1–32, 2010.

8 G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *International Conference on Very Large Databases (VLDB)*, pages 115–126, 2006.

9 G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *International Conference on Very Large Databases (VLDB)*, pages 998–1009, 2007.

10 C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems*, 5:1–24, 2009.

11 I. Boneva, R. Ciucanu, and S. Staworko. Schemas for unordered XML on a DIME. *Theory of Computing Systems*, 2014.

12 I. Boneva, J. Dusart, D. Fernández-Álvarez, and J. E. Labra Gayo. Semi automatic construction of shex and SHACL schemas. *CoRR*, abs/1907.10603, 2019. `arXiv:1907.10603`.

13 I. Boneva, J. Lozano, and S. Staworko. Relational to RDF data exchange in presence of a shape expression schema. In *Alberto Mendelzon International Workshop on Foundations of Data Management*, 2018.

14 A. Bonifati, S. Dumbrava, and H. Kondylakis. Graph summarization. *CoRR*, abs/2004.14794, 2020. `arXiv:2004.14794`.

15 P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding structure to unstructured data. In *International Conference on Database Theory (ICDT)*, pages 336–350, 1997.

16 S. Campinas, R. Delbru, and G. Tummarello. Efficiency and precision trade-offs in graph summary algorithms. In *International Database Engineering & Applications Symposium (IDEAS)*, pages 38–47, 2013.

17 J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67, 2007.

18 Š. Čebirić, F. Goasdoué, P. Guzewicz, and I. Manolescu. Compact summaries of rich heterogeneous graphs. Research Report RR-8920, INRIA Saclay ; Université Rennes 1, 2018.

19 Š. Čebirić, F. Goasdoué, H. Kondylakis, D. Kotzinos, I. Manolescu, G. Troullinou, and M. Zneika. Summarizing semantic graphs: a survey. *VLDB J.*, 28(3):295–327, 2019.

20 Š. Čebirić, F. Goasdoué, and I Manolescu. A framework for efficient representative summarization of RDF graphs. In *International Semantic Web Conference (ISWC)*, 2017.

21 B. Chidlovskii. Schema extraction from XML: A grammatical inference approach. In *Knowledge Representation Meets Databases (KRDB)*, volume 45, 2001.

22 R. Ciucanu and S. Staworko. Learning schemas for unordered XML. In *International Symposium on Database Programming Languages (DBPL)*, 2013.

23 W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *ACM SIGMOD International Conference on Management of Data*, pages 157–168, 2012.

24 M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *International Conference on Data Engineering (ICDE)*, pages 14–23, 1998.

25 M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: learning document type descriptors from XML document collections. *Data Mining and Knowledge Discovery*, 7(1):23–56, 2003.

26 F. Goasdoué, P. Guzewicz, and I. Manolescu. Incremental structural summarization of RDF graphs. In *International Conference on Extending Database Technology (EDBT)*, 2019.

27 F. Goasdoué, P. Guzewicz, and I. Manolescu. RDF graph summarization for first-sight structure discovery. *The VLDB Journal*, 29(5):1191–1218, 2020.

28 E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

29 R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *International Conference on Very Large Data Bases (VLDB)*, pages 436–445, 1997.

30 B. Groz, A. Lemay, S. Staworko, and P. Wieczorek. Inference of shape expression schemas typed RDF graphs. *CoRR*, abs/2107.04891, 2021. `arXiv:2107.04891`.

31 J. Heinz, A. Kasprzik, and T. Kötzing. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457:111–127, 2012.

32 A. Iana, S. Jung, P. Naeser, A. Birukou, S. Hertling, and H. Paulheim. Building a conference recommender system based on SciGraph and WikiCFP. In *Semantic Systems. The Power of AI and Knowledge Graphs*, pages 117–123, 2019.

33 S. Khatchadourian and M. Consens. ExpLOD: Summary-based exploration of interlinking and RDF usage in the linked open data cloud. In *Extended Semantic Web Conference (ESWC)*, pages 272–287, 2010.

34 H. Kondylakis, D. Kotzinos, and I. Manolescu. RDF graph summarization: principles, techniques and applications. In *International Conference on Extending Database Technology (EDBT)*, pages 433–436, 2019.

35 J. E. Labra Gayo, E. Prud'hommeaux, H. Solbrig, and J. M. Alvarez Rodriguez. Validating and describing linked data portals using RDF Shape Expressions. In *Workshop on Linked Data Quality*, 2015.

36 G. Laurence, A. Lemay, J. Niehren, S. Staworko, and M. Tommasi. Learning sequential tree-to-word transducers. In *Language and Automata Theory and Applications (LATA)*, pages 490–502, 2014.

37 H. Lbath, A. Bonifati, and R. Harmer. Schema inference for property graphs. In *International Conference on Extending Database Technology (EDBT)*, pages 499–504, 2021.

38 A. Lemay, S. Maneth, and J. Niehren. A learning algorithm for top-down XML transformations. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 285–296, 2010.

39 J.-K. Min, J.-Y. Ahn, and C.-W. Chung. Efficient extraction of schemas for XML documents. *Information Processing Letters*, 85(1):7–12, 2003.

40 S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *ACM SIGMOD International Conference on Management of Data*, pages 295–306, 1998.

41 S. Nestorov, J. D. Ullman, J. L. Wiener, and S. S. Chawathe. Representative objects: Concise representations of semistructured, hierarchial data. In *International Conference on Data Engineering (ICDE)*, pages 79–90, 1997.

42 E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig. Shape Expressions: An RDF validation and transformation language. In *International Conference on Semantic Systems*, 2015.

43 J. T. Reese, D. Unni, Callahan T. J., L. Cappelletti, V. Ravanmehr, S. Carbon, K. A. Shefchek, B. M. Good, J. P. Balhoff, T. Fontana, H. Blau, N. Matentzoglu, N. L. Harris, M. C. Munoz-Torres, M. A. Haendel, P. N. Robinson, M. P. Joachimiak, and C. J. Mungall. KG-COVID-19: A framework to produce customized knowledge graphs for COVID-19 response. *Patterns*, 2(1):100155, 2021.

44 A. Schätzle, A. Neu, G. Lausen, and M. Przyjaciel-Zablocki. Large-scale bisimulation of RDF graphs. In *Workshop on Semantic Web Information Management (SWIM)*, pages 1–8, 2013.

45 J. Sequeda, S. H. Tirmizi, Ó. Corcho, and D. P. Miranker. Survey of directly mapping SQL databases to the Semantic Web. *Knowledge Engineering Review*, 26(4):445–486, 2011.

46 J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to RDF and OWL. In *International Conference on World Wide Web (WWW)*, pages 649–658, 2012.

47 S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig. Complexity and expressiveness of ShEx for RDF. In *International Conference on Database Theory (ICDT)*, pages 195–211, 2015.

48 S. Staworko and P. Wieczorek. Learning twig and path queries. In *International Conference on Database Theory (ICDT)*, pages 140–154, 2012.

49 S. Staworko and P. Wieczorek. Containment of shape expression schemas for RDF. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 303–319, 2019.

50 F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *International Conference on World Wide Web (WWW)*, pages 697–0706. Association for Computing Machinery, 2007.

51 TPC. TPC benchmarks. URL: `http://www.tpc.org/`.

52 T. Tran, G. Ladwig, and S. Rudolph. Managing structured and semistructured RDF data using structure indexes. *Transactions on Knowledge and Data Engineering*, 25(9):2076–2089, 2013.

53 Y. Tsuboi and N. Suzuki. An algorithm for extracting shape expression schemas from graphs. In *ACM Symposium on Document Engineering (DocEng)*, pages 1–4, 2019.

54 W3C. A direct mapping of relational data to RDF, 2012. URL: `http://www.w3.org/TR/rdb-direct-mapping/`.

55 W3C. Shape expressions schemas, 2013. URL: `http://www.w3.org/2013/ShEx/Primer`.

56 H. Zhang, Y. Duan, X. Yuan, and Y. Zhang. Assg: adaptive structural summary for RDF graph data. In *International Semantic Web Conference (ISWC) Posters & Demonstrations Track*, pages 233–236, 2014.

Expressiveness of SHACL Features

Bart Bogaerts
Vrije Universiteit Brussel, Belgium

Maxime Jakubowski
Hasselt University, Diepenbeek, Belgium

Jan Van den Bussche
Hasselt University, Diepenbeek, Belgium

Abstract

SHACL is a W3C-proposed schema language for expressing structural constraints on RDF graphs. Recent work on formalizing this language has revealed a striking relationship to description logics. SHACL expressions can use four fundamental features that are not so common in description logics. These features are zero-or-one path expressions; equality tests; disjointness tests; and closure constraints. Moreover, SHACL is peculiar in allowing only a restricted form of expressions (so-called targets) on the left-hand side of inclusion constraints.

The goal of this paper is to obtain a clear picture of the impact and expressiveness of these features and restrictions. We show that each of the four features is primitive: using the feature, one can express boolean queries that are not expressible without using the feature. We also show that the restriction that SHACL imposes on allowed targets is inessential, as long as closure constraints are not used.

2012 ACM Subject Classification Information systems → Semantic web description languages; Information systems → Query languages; Theory of computation → Description logics; Theory of computation → Finite Model Theory

Keywords and phrases Expressive power, schema languages

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.15

Funding Supported by AI Research Flanders.

Acknowledgements We thank Dörthe Arndt for inspiring discussions.

1 Introduction

On the Web, the Resource Description Framework (RDF [16]) is a standard format for representing knowledge and publishing data. RDF represents information in the form of directed graphs, where labeled edges indicate properties of nodes. To facilitate more effective access and exchange, it is important for a consumer of an RDF graph to know what properties to expect, or, more generally, to be able to rely on certain structural constraints that the graph is guaranteed to satisfy. We therefore need a declarative language in which such constraints can be expressed formally. In database terms, we need a schema language.

Two prominent proposals in this vein have been ShEx [6] and SHACL [18]. Both approaches use formulas that express the presence or absence of certain properties of a node or its neighbors in the graph. Such formulas are called "shapes." When we evaluate a shape on a node, that node is called the "focus node." Some examples of shapes, expressed for now in English, could be the following:[1]

[1] In real RDF, names of properties and nodes must conform to IRI syntax, but in this paper, to avoid clutter, we take the liberty to use simple names.

1. "The focus node has a phone property, but no email."
2. "The focus node has at least five incoming managed-by edges."
3. "Through a path of friend edges, the focus node can reach a node with a CEO-of edge to the node Apple."
4. "The focus node has at least one colleague who is also a friend."
5. "The focus node has no other properties than name, address, or birthdate."

In this paper, we look deeper into SHACL, the language recommended by the World Wide Web Consortium. We do not use the actual SHACL syntax, but work with the elegant formalization proposed by Corman, Reutter and Savkovic [9], and used in subsequent works by several authors [2, 13, 15]. That formalization reveals a striking similarity between shapes on the one hand, and concepts, familiar from description logics [4], on the other hand. The similarity between SHACL and description logics runs even deeper when we account for targeting, which is the actual mechanism to express constraints on an RDF graph using shapes.

Specifically, a non-recursive *shape schema*[2] is essentially a finite list of shapes, where each shape ϕ is additionally associated with a target query q. An RDF graph G is said to conform to such a schema if for every target–shape combination (q, ϕ), and every node v returned by q on G, we have that v satisfies ϕ in G. Let us see some examples of target–shape pairs, still expressed in English:

6. "Every node of type Person has an email or phone property." Here, the target query returns all nodes with an edge labeled type to node `Person`; the shape checks that the focus node has an email or phone property.
7. "Different nodes never have the same email." Here the target query returns all nodes with an incoming email edge, and the shape checks that the focus node does not have two or more incoming email edges.
8. "Every mathematician has a finite Erdős number." Here the target query returns all nodes of type Mathematician, and the shape checks that the focus node can reach the node Erdős by a path that matches the regular expression $(\mathsf{author}^{-}/\mathsf{author})^{*}$. Here, the minus superscript denotes an inverse edge.

Now interestingly, and apparent in the examples 6–8, the target queries considered for this purpose in SHACL, as well as in ShEx, actually correspond to simple cases of shapes. It is then only a small step to consider *generalized* shape schemas as finite sets of inclusion statements of the form $\phi_1 \subseteq \phi_2$, where ϕ_1 and ϕ_2 are shapes. Since, as noted above, shapes correspond to concepts, we thus see that shape schemas correspond to TBoxes in description logics.

We stress that the task we are focusing on in this paper is checking conformance of RDF graphs against shape schemas. Every shape schema $\mathcal{S}$ defines a decision problem: given an RDF graph G, check whether G conforms to $\mathcal{S}$. In database terms, we are processing a *boolean query* on a graph database. In description logic terms, this amounts to *model checking* of a TBox: given an interpretation, check whether it satisfies the TBox. Thus our focus is a bit different from that of typical applications of description logics. There, facts are declared in ABoxes, which should not be confused with interpretations. The focus is then on higher reasoning tasks, such as checking satisfiability of an ABox and a TBox, or deciding logical entailment.

[2] Real SHACL uses the term *shapes graph* instead of shape schema.

Given the above context, let us now look in more detail at the logical constructs that can be used to build shapes. Some of these constructs are well known concept constructors from expressive description logics [7]: the boolean connectives; constants; qualified number restriction (a combination of existential quantification and counting); and regular path expressions with inverse. To illustrate, example shapes 1–3 are expressible as follows:

1. $\geq_1 \mathsf{phone}.\top \wedge \neg \geq_1 \mathsf{email}.\top$. This uses qualified number restriction with count 1 (so essentially existential quantification), conjunction, and negation; $\top$ stands for true.
2. $\geq_5 \mathsf{managed\text{-}by}^-.\top$. This uses counting to 5, and inverse.
3. $\geq_1 \mathsf{friend}^*/\mathsf{CEO\text{-}of}.\{\mathsf{Apple}\}$. This uses a regular path expression and the constant Apple.

However, SHACL also has four specific logical features that are less common:

Equality, disjointness: The shape $eq(E, r)$, for a path expression E and a property r, tests equality of the sets of nodes reachable from the focus node by an r-edge on the one hand, and by an E-path on the other hand. A similar shape $disj(E, r)$ tests disjointness of the two sets. To illustrate, example shape 4 is expressed as $\neg disj(\mathsf{colleague}, \mathsf{friend})$.

Closure constraints: RDF graphs to be checked for conformance against some shape schema need not obey some fixed vocabulary. Thus SHACL provides shapes of the form $closed(R)$, with R a finite set of properties, expressing that the focus node has no properties other than those in R. This was already illustrated as example shape 5, with $R = \{\mathsf{name}, \mathsf{address}, \mathsf{birthdate}\}$.

Zero-or-one paths: If E is a path expression, then the path expression $E?$, evaluated in a focus node v, yields the set of nodes reachable from v by an E-path, plus v itself. This constructor, which is also present in the RDF query language SPARQL, is a very special case of *tests* in the logic PDL [7], where $E?$ would be written as $E \cup \top?$. For example, in a function call graph as used in software engineering, the shape $\geq_5 \mathsf{calls}?.\top$ expresses that the focus node calls at least *four* functions other than itself. Interestingly, zero-or-one paths allow the expression of self-restriction, a feature introduced in the logic SROIQ [11] and adopted in the Web ontology language [14]. For example, staying with the function calls, to express in SROIQ that the focus node calls itself, one can write $\exists \mathsf{calls}.\mathsf{Self}$. As a SHACL shape, we can write $eq(\mathsf{calls}?, \mathsf{calls})$.

Our goal in this paper is to clarify the impact of the above four features on the expressiveness of SHACL as a language for boolean queries on graph databases. Thereto, we offer the following contributions.

- We show that each of the four features is primitive in a strong sense. Specifically, for each feature, we exhibit a boolean query Q such that Q is expressible by a single target–shape pair, using only the feature and the basic constructs; however, Q is not expressible by any generalized shape schema when the feature is disallowed.
- We also clarify the significance of the restriction that SHACL puts on allowed targets. We observe that as long as closure constraints are not used, the restriction is actually insignificant. Any generalized shape schema, allowing arbitrary but closure-free shapes on the left-hand sides of the inclusion statements, can be equivalently written as a shape schema with only targets on the left-hand sides. However, allowing closure constraints on the left-hand side strictly adds expressive power.
- Our results continue to hold when the definition of *recursive* shapes is allowed, provided that recursion through negation is stratified.

This paper is organized as follows. Section 2 presents clean formal definitions of non-recursive shape schemas, building on and inspired by the work of Andreşel, Corman, et al. cited above. Section 3 presents our results. Section 4 presents the extension to stratified recursion. Section 5 offers concluding remarks.

2 Shape schemas

In this section we define shapes, RDF graphs, shape schemas, and the conformance of RDF graphs to shape schemas. Perhaps curiously to those familiar with SHACL, our treatment for now omits *shape names*. Shape names are redundant as far as expressive power is concerned, as long as we are in a non-recursive setting, because shape name definitions can then always be unfolded. Indeed, for clarity of exposition, we have chosen to work first with non-recursive shape schemas. Section 4 then presents the extension to recursion (and introduces shape names in the process). We point out that the W3C SHACL recommendation only considers non-recursive shape schemas.

Node and property names

From the outset we assume two disjoint, infinite universes N and P of *node names* and *property names*, respectively.[3]

2.1 Shapes

We define *path expressions* E and *shapes* ϕ by the following grammar:

$$E ::= p \mid p^- \mid E \cup E \mid E/E \mid E? \mid E^*$$
$$\phi ::= \top \mid \{c\} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \geq_n E.\phi \mid \mathit{eq}(E,p) \mid \mathit{disj}(E,p) \mid \mathit{closed}(R)$$

Here, p and c stand for property names and node names, respectively; n stands for nonzero natural numbers; and R stands for finite sets of property names. A node name c is also referred to as a *constant*.

Abbreviation: We will use $\exists E.\phi$ as an abbreviation for $\geq_1 E.\phi$.

▶ Remark 1. Real SHACL supports some further shapes which have to do with tests on IRI constants and literals, as well as comparisons on numerical values and language tags. Like other work on the formal aspects of SHACL, we abstract these away, as many questions are already interesting without these features.

▶ Remark 2. Universal quantification $\forall E.\phi$ could be introduced as an abbreviation for $\neg\exists E.\neg\phi$. Likewise, $\leq_n E.\phi$ may be used as an abbreviation for $\neg \geq_{n+1} E.\phi$.

A *vocabulary* Σ is a subset of $N \cup P$. A path expression is said to be *over* Σ if it only uses property names from Σ. Likewise, a shape is over Σ if it only uses constants from Σ and path expressions over Σ.

Following common practice in logic, shapes are evaluated in *interpretations*. We recall the familiar definition of an interpretation. Let Σ be a vocabulary. An interpretation I over Σ consists of

- a set Δ^I, called the *domain* of I;
- for each constant $c \in \Sigma$, an element $[\![c]\!]^I \in \Delta^I$; and
- for each property name $p \in \Sigma$, a binary relation $[\![p]\!]^I$ on Δ^I.

The semantics of shapes is now defined as follows.

[3] In practice, node names and property names are IRIs [16], so the disjointness assumption would not hold. However, this assumption is only made for simplicity of notation; it can be avoided if we make our notation for vocabularies and interpretations (see below) more complicated.

Table 1 Semantics of path expressions.

E	$[\![E]\!]^I$
p^-	$\{(a,b) \mid (b,a) \in [\![p]\!]^I\}$
$E_1 \cup E_2$	$[\![E_1]\!]^I \cup [\![E_2]\!]^I$
E_1/E_2	$\{(a,b) \mid \exists x : (a,x) \in [\![E_1]\!]^I \wedge (x,b) \in [\![E_2]\!]^I\}$
$E?$	$[\![E]\!]^I \cup \{(x,x) \mid x \in \Delta^I\}$
E^*	the reflexive-transitive closure of $[\![E]\!]^I$

Table 2 Conditions for conformance of a node to a shape.

ϕ	$I, a \vDash \phi$ if:
$\{c\}$	$a = [\![c]\!]^I$
$\geq_n E.\psi$	$\sharp\{b \in [\![E]\!]^I(a) \mid I, b \vDash \psi\} \geq n$
$eq(E,p)$	the sets $[\![E]\!]^I(a)$ and $[\![p]\!]^I(a)$ are equal
$disj(E,p)$	the sets $[\![E]\!]^I(a)$ and $[\![p]\!]^I(a)$ are disjoint
$closed(R)$	$[\![p]\!]^I(a)$ is empty for each $p \in \Sigma - R$

- On any interpretation I as above, every path expression E over Σ evaluates to a binary relation $[\![E]\!]^I$ on Δ^I, defined in Table 1.
- Now for any shape ϕ over Σ and any element $a \in \Delta^I$, we define when a *conforms to* ϕ *in* I, denoted by $I, a \vDash \phi$. For the boolean operators $\top$ (true), $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), the definition is obvious. For the other constructs, the definition is given in Table 2, taking note of the following:
  - We use the notation $R(x)$, for a binary relation R, to denote the set $\{y \mid (x,y) \in R\}$. We apply this notation to the case where R is of the form $[\![E]\!]^I$.
  - We also use the notation $\sharp X$ for the cardinality of a set X.
- For a shape ϕ and interpretation I, the notation

  $$[\![\phi]\!]^I := \{a \in \Delta^I \mid I, a \vDash \phi\}$$

  will be convenient.

2.2 Graphs and their interpretation

It may appear that a shape $closed(R)$ is simply expressible as the conjunction of $\neg\exists p.\top$ for $p \in \Sigma - R$. However, since shapes must be finite formulas, this only works if Σ is finite. In practice, shapes can be evaluated over arbitrary RDF graphs, which can involve arbitrary property names (and node names), not limited to a finite vocabulary that is fixed in advance.

Formally, we define a *graph* as a finite set of triples of the form (a,p,b), where p is a property name and a and b are (not necessarily distinct) node names. We refer to the node names appearing in a graph G simply as the *nodes* of G; the set of nodes of G is denoted by N_G. A pair (a,b) with $(a,p,b) \in G$ is referred to as an *edge*, or a *p-edge*, in G.

We now canonically view any graph G as an interpretation over the *full* vocabulary $N \cup P$ as follows:

- Δ^G equals N (the universe of all node names).
- $[\![c]\!]^G$ equals c itself, for every node name c.
- $[\![p]\!]^G$ equals the set of p-edges in G, for every property name p.

Note since graphs are finite, $[\![p]\!]^G$ will be empty for all but a finite number of p's.

Given this canonical interpretation, path expressions and shapes obtain a semantics on all graphs G. Thus for any path expression E, the binary relation $[\![E]\!]^G$ on N is well-defined; for any shape ϕ and $a \in N$, it is well-defined whether or not $G, a \vDash \phi$; and we can also use the notation $[\![\phi]\!]^G$.

▶ Remark 3. Since a graph is considered to be an interpretation with the infinite domain N, it may not be immediately clear that shapes can be effectively evaluated over graphs. Adapting well-known methods, however, we can reduce to a finite domain over a finite vocabulary [1, Theorem 5.6.1], [3, 12]. Formally, let ϕ be a shape and let G be a graph. Recall that N_G denotes the set of nodes of G; similarly, let P_G be the set of property names appearing in G. Let C be the set of constants mentioned in ϕ. We can then form the finite vocabulary $\Sigma = N_G \cup C \cup P_G$. Now define the interpretation I over Σ as follows:

- $\Delta^I = N_G \cup C \cup \{\star\}$, where $\star$ is an element not in N;
- $[\![c]\!]^I = c$ for each node name $c \in \Sigma$;
- $[\![p]\!]^I = [\![p]\!]^G$ for each property name $p \in \Sigma$.

Note that no constant symbol names $\star$ in I. Then for every $x \in N_G \cup C$, one can show that $x \in [\![\phi]\!]^G$ if and only if $x \in [\![\phi]\!]^I$. For all other node names x, one can show that $x \in [\![\phi]\!]^G$ if and only if $\star \in [\![\phi]\!]^I$.

In our companion paper [5] we argue why using the infinite domain N is the most natural approach.

2.3 Targets and shape schemas

SHACL identifies four special forms of shapes and calls them *targets*:

Node targets: $\{c\}$ for any constant c.

Class-based targets: $\exists \mathsf{type}/\mathsf{subclass}^*.\{c\}$ for any constant c. Here, type and subclass represent distinguished IRIs from the RDF Schema vocabulary [16].

Subjects-of targets: $\exists p.\top$ for any property name p.

Objects-of targets: $\exists p^-.\top$ for any property name p.

We now define a *generalized shape schema* (or shape schema for short) as a finite set of inclusion statements, where an inclusion statement is of the form $\phi_1 \subseteq \phi_2$, with ϕ_1 and ϕ_2 shapes. A *target-based shape schema* is a shape schema that only uses targets, as defined above, on the left-hand sides of its inclusion statements. This restriction corresponds to the shape schemas considered in real SHACL.

As already explained in the Introduction, a graph G *conforms* to a shape schema $\mathcal{S}$, denoted by $G \vDash \mathcal{S}$, if $[\![\phi_1]\!]^G$ is a subset of $[\![\phi_2]\!]^G$, for every statement $\phi_1 \subseteq \phi_2$ in $\mathcal{S}$.

Thus, any shape schema $\mathcal{S}$ defines the class of graphs that conform to it. We denote this class of graphs by

$$[\![\mathcal{S}]\!] := \{\text{graph } G \mid G \vDash \mathcal{S}\}.$$

Accordingly, two shape schemas $\mathcal{S}_1$ and $\mathcal{S}_2$ are said to be *equivalent* if $[\![\mathcal{S}_1]\!] = [\![\mathcal{S}_2]\!]$.

3 Expressiveness of SHACL features

When a complicated but influential new tool is proposed in the community, in our case SHACL, we feel it is important to have a solid understanding of its design. Concretely, as motivated in the Introduction, our goal is to obtain a clear picture of the relative expressiveness of the features *eq*, *disj*, *closed*, and ?. Our methodology is as follows.

A *feature set* F is a subset of $\{eq, disj, closed, ?\}$. The set of all shape schemas using only features from F, besides the standard constructs, is denoted by $\mathcal{L}(F)$. In particular, shape schemas in $\mathcal{L}(\emptyset)$ use only the standard constructs and none of the four features. Specifically, they only involve shapes built from boolean connectives, constants, and qualified number restrictions, with path expressions built from the standard operators union, composition, and Kleene star.

We say that feature set F_1 is *subsumed* by feature set F_2, denoted by $F_1 \preceq F_2$, if every shape schema in $\mathcal{L}(F_1)$ is equivalent to some shape schema in $\mathcal{L}(F_2)$. As it will turn out,

$$F_1 \preceq F_2 \quad \Leftrightarrow \quad F_1 \subseteq F_2, \tag{$*$}$$

or intuitively, "every feature counts." Note that the implication from right to left is trivial, but the other direction is by no means clear from the outset.

More specifically, for every feature, we introduce a class of graphs, as follows. In what follows we fix some property name r.

Equality: Q_{eq} is the class of graphs where all r-edges are symmetric. Note that Q_{eq} is definable in $\mathcal{L}(eq)$ by the single, target-based, inclusion statement $\exists r.\top \subseteq eq(r^-, r)$.

Disjointness: Q_{disj} is the class of graphs where all nodes with an outgoing r-edge have at least one symmetric r-edge. This time, Q_{disj} is definable in $\mathcal{L}(disj)$, by the single, target-based, inclusion statement $\exists r.\top \subseteq \neg disj(r^-, r)$.

Closure: Q_{closed} is the class of graphs where for all nodes with an outgoing r-edge, all outgoing edges have label r. Again Q_{closed} is definable in $\mathcal{L}(closed)$ by the single, target-based, inclusion statement $\exists r.\top \subseteq closed(r)$.

Zero-or-one path: Finally, $Q_?$ is the class of graphs where all nodes with an outgoing r-edge have at least three outgoing r-edges *not* to themselves (i.e., not counting possible self-loops). As expected $Q_?$ is definable in $\mathcal{L}(?)$ by the single, target-based, inclusion statement $\exists r.\top \subseteq \geq_4 r?.\top$.

We establish the following theorem, from which the above equivalence $(*)$ immediately follows:

▶ **Theorem 4.** *Let $X \in \{eq, disj, closed, ?\}$ and let F be a feature set with $X \notin F$. Then Q_X is not definable in $\mathcal{L}(F)$.*

3.1 Equality, disjointness, and zero-or-one path

For $X = closed$, Theorem 4 is proven differently than for the other three features. We deal with *closed* in the next subsection. Here, we deal with the remaining features through the following concrete result, illustrated in Figure 1:

▶ **Proposition 5.** *Let $X = eq$, $disj$, or $?$. Let Σ be a finite vocabulary including r, and let m be a nonzero natural number. There exist two graphs G and G' with the following properties:*

1. *G' belongs to Q_X, but G does not.*
2. *For every shape ϕ over Σ such that ϕ does not use X, and ϕ counts to at most m, we have*

$$[\![\phi]\!]^G = [\![\phi]\!]^{G'}.$$

Here, "counting to at most m" means that all quantifiers $\geq_n$ used in ϕ satisfy $n \leq m$.

To see that Proposition 5 indeed establishes Theorem 4 for the three features under consideration, we use the notion of *validation shape* of a shape schema. This shape evaluates to the set of all nodes that violate the schema. Thus, the validation shape is an abstraction

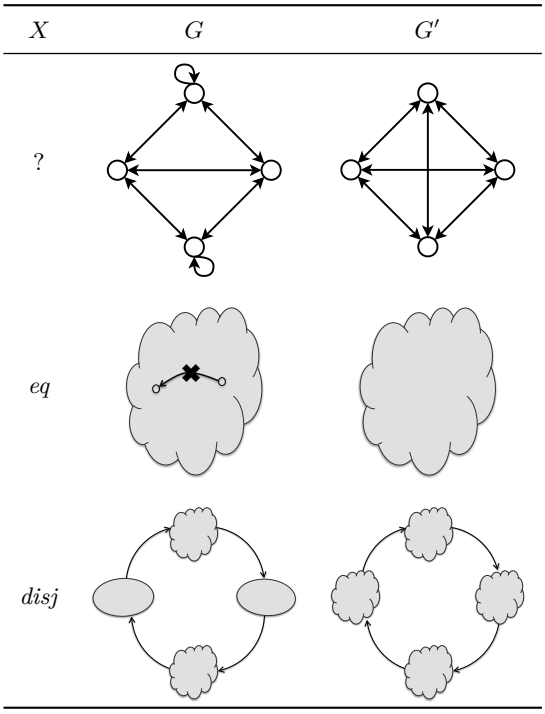


Figure 1 Graphs used to prove Proposition 5. The nodes are taken outside Σ. For $X = eq$, the cloud shown for G' represents a complete directed graph on $m + 1$ nodes, with self-loops, and G is the same graph with one directed edge removed. For $X = disj$, in the picture for G, each cloud again stands for a complete graph, but this time on $M = \max(m, 3)$ nodes, and without the self-loops. Each oval stands for a set of M separate nodes. An arrow from one blob to the next means that every node of the first blob has a directed edge to every node of the next blob. So, G is a directed 4-cycle of alternating clouds and ovals, and G' is a directed 4-cycle of clouds.

of the "validation report" in SHACL [18]: a graph conforms to a schema if and only if the validation shape evaluates to the empty set. The validation shape can be formally constructed as the disjunction of $\phi_1 \wedge \neg\phi_2$ for all statements $\phi_1 \subseteq \phi_2$ in the schema.

Now consider a shape schema $\mathcal{S}$ not using feature X. Let m be the maximum count used in shapes in $\mathcal{S}$, and let Σ' be the set of constants and property names mentioned in $\mathcal{S}$. Now given $\Sigma = \Sigma' \cup \{r\}$ and m, let G and G' be the two graphs exhibited by the Proposition, and let ϕ be the validation shape for $\mathcal{S}$. Then ϕ will evaluate to the same result on G and G'. However, for $\mathcal{S}$ to define Q_X, validation would have to return the empty set on G' but a nonempty set on G. We conclude that $\mathcal{S}$ does not define Q_X.

Proof of Proposition 5

We present here the proof for $X = disj$. The general strategy is to first characterize the behavior of path expressions on G and G'. Then the Proposition is proven with a stronger induction hypothesis, to allow the induction to carry through. A similar strategy is followed in the proofs for $X = eq$ and $X = ?$.

We begin by defining the graphs G and G' more formally.

▶ **Definition 6** ($G_{disj}(\Sigma, m)$). *Let Σ be a finite vocabulary including r, and let m be a natural number. We define the graph $G_{disj}(\Sigma, m)$ over the set of property names in Σ as follows. Let $M = \max(m, 3)$. There are $4M$ nodes in the graph, which are chosen outside of Σ. We*

denote these nodes by x_i^j *for* $i = 1, 2, 3, 4$ *and* $j = 1, \ldots, M$*. (In the description that follows, subscripts range from* 1 *to* 4 *and superscripts range from* 1 *to* M*.) For each property name* p *in* Σ*, the graph has the same set of* p*-edges. There is an edge from* x_i^j *to* $x_{i \bmod 4+1}^{j'}$ *for every* i*,* j *and* j'*. Moreover, if* i *is 2 or 4, there is an edge from* x_i^j *to* $x_i^{j'}$ *for all* $j \neq j'$*.*

Thus, in Figure 1, bottom left, one can think of the left oval as the set of nodes x_1^j; the top cloud as the set of nodes x_2^j; and so on. We call the nodes x_i^j with $i = 2, 4$ the *even nodes*, and the nodes x_i^j with $i = 1, 3$ the *odd nodes*.

▶ **Definition 7** ($G'_{disj}(\Sigma, m)$)**.** *We define the graph* $G'_{disj}(\Sigma, m)$ *in the same way as* $G_{disj}(\Sigma, m)$ *except that there is an edge from* x_i^j *to* $x_i^{j'}$ *for all* i *and* $j \neq j'$ *(not only for* i *even).*

We characterize the behavior of path expressions on the graph $G_{disj}(\Sigma, m)$ as follows.

▶ **Lemma 8.** *Let* G *be* $G_{disj}(\Sigma, m)$*. Call a path expression* simple *if it is a union of property names. Let* E *be a non-simple path expression over* Σ*. The following three statements hold:*

1. **A.** *for all even nodes* v *of* G*, we have* $[\![E]\!]^G(v) \supseteq [\![r]\!]^G(v)$*; or*
   B. *for all even nodes* v *of* G*, we have* $[\![E]\!]^G(v) \supseteq [\![r^-]\!]^G(v)$*.*
2. **C.** *for all odd nodes* v *of* G*, we have* $[\![E]\!]^G(v) \supseteq [\![r]\!]^G(v)$*; or*
   D. *for all odd nodes* v *of* G*, we have* $[\![E]\!]^G(v) \supseteq [\![r^-]\!]^G(v)$*.*
3. *For all nodes* v *of* G*, we have* $[\![E]\!]^G(v) - [\![r]\!]^G(v) \neq \emptyset$*.*

Proof. We use the notation $X_i = \{x_i^1, \ldots, x_i^M\}$ for the ith blob of nodes. We also use the notations $next(1) = 2$; $next(2) = 3$; $next(3) = 4$; $next(4) = 1$; $prev(4) = 3$; $prev(3) = 2$; $prev(2) = 1$; $prev(1) = 4$. Thus $next(i)$ indicates the next blob in the cycle, and $prev(i)$ the previous.

The proof is by induction on the structure of E. If E is a property name, E is simple so the claim is trivial. If E is of the form p^-, cases B and D are clear and we only need to verify the third statement. That holds because for any i, if $v \in X_i$, then $[\![p^-]\!]^G(v) \supseteq X_{prev(i)}$ and clearly $X_{prev(i)} - [\![r]\!]^G(v) \neq \emptyset$. We next consider the inductive cases.

First, assume E is of the form $E_1 \cup E_2$. When at least one of E_1 and E_2 is not simple, the three statements immediately follow by induction, since $[\![E]\!]^G \supseteq [\![E_1]\!]^G$ and $[\![E]\!]^G \supseteq [\![E_2]\!]^G$. If E_1 and E_2 are simple, then E is simple and the claim is trivial.

Next, assume E is of the form $E_1?$ or E_1^*. If E_1 is not simple, the three statements follow immediately by induction, since $[\![E]\!]^G \supseteq [\![E_1]\!]^G$. If E_1 is simple, cases A and C clearly hold for E, so we only need to verify the third statement. That holds because, by the form of E, every node v is in $[\![E]\!]^G(v)$, but not in $[\![r]\!]^G(v)$, as G does not have any self-loops.

Finally, assume E is of the form E_1/E_2. Note that if E_1 or E_2 is simple, clearly cases A and C apply to them. The argument that follows will therefore also apply when E_1 or E_2 is simple. We will be careful not to apply the induction hypothesis for the third statement to E_1 and E_2.

We first focus on the even nodes, and show the first and the third statement. We distinguish two cases.

- If case A applies to E_2, then we show that case A also applies to E. Let $v \in X_i$ be an even node. We verify the following two inclusions:
  - $[\![E]\!]^G(v) \supseteq X_i$. Let $u \in X_i$. If $u \neq v$, choose a third node $w \in X_i$. Since X_i is a clique, $(v, w) \in [\![E_1]\!]^G$ regardless of whether case A or B applies to E_1. By case A for E_2, we also have $(w, u) \in [\![E_2]\!]^G$, whence $u \in [\![E]\!]^G(v)$ as desired. If $u = v$, we similarly have $(v, w) \in [\![E_1]\!]^G$ and $(w, u) \in [\![E_2]\!]^G$ as desired.

- $[\![E]\!]^G(v) \supseteq X_{next(i)}$. Let $u \in X_{next(i)}$ and choose $w \neq v \in X_i$. Regardless of whether case A or B applies to E_1, we have $(v, w) \in [\![E_1]\!]^G$. By case A for E_2, we also have $(w, u) \in [\![E_2]\!]^G$, whence $u \in [\![E]\!]^G(v)$ as desired.

We conclude that $[\![E]\!]^G(v) \supseteq X_i \cup X_{next(i)} \supseteq [\![r]\!]^G$ as desired.

- If case B applies to E_2, then we show that case B also applies to E. This is analogous to the previous case, now verifying that $[\![E]\!]^G(v) \supseteq X_i \cup X_{prev(i)}$.

In both cases, the third statement now follows for even nodes v. Indeed, $v \in X_i \subseteq [\![E]\!]^G(v)$ but $v \notin [\![r]\!]^G(v)$.

We next focus on the odd nodes, and show the second and the third statement. We again consider two cases.

- If case C applies to E_1, then we show that case C also applies to E. Let $v \in X_i$ be an odd node. Note that $[\![r]\!]^G(v) = X_{next(i)}$. To verify that $[\![E]\!]^G(v) \supseteq X_{next(i)}$, let $u \in X_{next(i)}$. Then u is even. Choose $w \neq u \in X_{next(i)}$. Since case C applies to E_1, we have $(v, w) \in [\![E_1]\!]^G$. Moreover, since $X_{next(i)}$ is a clique, $(w, u) \in [\![E_2]\!]^G$ regardless of whether case A or B applies to E_2. We obtain $(v, u) \in [\![E]\!]^G$ as desired.

  We also verify the third statement for odd nodes in this case. We distinguish two further cases.
  - If case A applies to E_2, any node $u \in X_{next(next(i))}$ belongs to $[\![E]\!]^G(v)$, and clearly these u are not in $X_{next(i)} = [\![r]\!]^G(v)$.
  - If case B applies to E_2, then, since X_i is a clique, any node $u \in X_i$ belongs to $[\![E]\!]^G(v)$, and again these u are not in $X_{next(i)}$.
- If case D applies to E_1, then we show that case D also applies to E. This is analogous to the previous case, now verifying that $[\![E]\!]^G(v) \supseteq X_{prev(i)}$. In this case the third statement for odd nodes is clear, as clearly $X_{prev(i)} - X_{next(i)} \neq \emptyset$. ◀

We similarly characterize the behavior of path expressions on the other graph. The characterization is simpler to state and simpler to verify, due to the homogeneous nature of the graph $G'_{disj}(\Sigma, m)$. We omit the proof.

▶ **Lemma 9.** *Let G' be $G'_{disj}(\Sigma, m)$ and let E be a non-simple path expression over Σ. The following statements hold:*

1. $[\![E]\!]^{G'} \supseteq [\![r]\!]^{G'}$ *or* $[\![E]\!]^{G'} \supseteq [\![r^-]\!]^{G'}$.
2. *For all nodes v of G', we have $[\![E]\!]^{G'}(v) - [\![r]\!]^{G'}(v) \neq \emptyset$.*

We also need the following Definition and Lemma, detailing how path expressions can behave on the nodes outside a graph.

▶ **Definition 10** (Safety). *We define the* safety *of a path expression E inductively as follows:*

- *If E is p or p^-, then E is safe.*
- *If E is $E_1 \cup E_2$, then E is safe only if E_1 and E_2 are safe.*
- *If E is E_1/E_2, then E is safe if E_1 or E_2 is safe.*
- *If E is E_1^* or $E?$, then E is not safe.*

▶ **Lemma 11.** *Let E be a path expression and let G be a graph.*

- *If E is safe, then $[\![E]\!]^G \subseteq N_G \times N_G$.*
- *If E is not safe, then $[\![E]\!]^G = ([\![E]\!]^G \cap N_G \times N_G) \cup \{(a, a) \mid a \in N - N_G\}$.*

We are now ready to prove the non-obvious part of Proposition 5, in the following version.

▶ **Proposition 5 for $X = disj$, stronger version.** *Let V be the common set of nodes of the graphs $G = G_{disj}(\Sigma, m)$ and $G' = G'_{disj}(\Sigma, m)$. Let ϕ be a shape over Σ that does not use disj, and that maximally counts to m. Then either $[\![\phi]\!]^G \cap V = \emptyset$ or $[\![\phi]\!]^G \supseteq V$. Moreover, $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$.*

This is proven by induction on the structure of ϕ. Let H be G or G'. If ϕ is $\top$, then $[\![\top]\!]^H = N \supseteq V$. If ϕ is $\{c\}$, then $[\![\{c\}]\!]^H = \{c\} \subseteq \Sigma$ and we know that $\Sigma \cap V = \emptyset$. We next consider the inductive cases. The cases for the boolean connectives follow readily by induction.

Now assume ϕ is of the form $\geq_n E.\phi_1$. By induction, there are two possibilities for ϕ_1:

- If $[\![\phi_1]\!]^H \cap V = \emptyset$, then also $[\![\phi]\!]^H \cap V = \emptyset$ since path expressions can only reach nodes in some graph from nodes in that graph.
- If $[\![\phi_1]\!]^H \supseteq V$, to show that $[\![\phi]\!]^H \supseteq V$, it suffices to show that $\sharp[\![E]\!]^H(v) \geq n$ for all $v \in V$. By Lemmas 8 and 9 we know that $[\![E]\!]^H(v)$ contains $[\![r]\!]^H(v)$ or $[\![r^-]\!]^H(v)$. Inspecting H, we see that each of these sets has at least $\max(3, m) \geq n$ elements, as desired.

In both cases we still need to show that $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$. We already showed that $[\![\phi]\!]^G \supseteq V$ and $[\![\phi]\!]^{G'} \supseteq V$, or $[\![\phi]\!]^G \cap V = \emptyset$ and $[\![\phi]\!]^{G'} \cap V = \emptyset$. Therefore, towards a proof of the equality, we only need to consider the node names not in V.

For the inclusion from left to right, take $x \in [\![\phi]\!]^G - V$. Since $G, x \vDash \phi$, there exists y_1, $\ldots$, y_n such that $(x, y_i) \in [\![E]\!]^G$ and $G, y_i \vDash \phi_1$ for $i = 1, \ldots, n$. However, since $x \notin V$, by Lemma 11, all y_i must equal x. Hence, $n = 1$ and $(x, x) \in [\![E]\!]^G$ and $G, x \vDash \phi_1$. Then again by the same Lemma, $(x, x) \in [\![E]\!]^{G'}$, since G and G' have the same set of nodes V. Moreover, by induction, $G', x \vDash \phi_1$. We conclude that $G', x \vDash \phi$ as desired. The inclusion from right to left is argued symmetrically.

Next assume ϕ is of the form $eq(E, p)$. If E is simple then $[\![E]\!]^H = [\![p]\!]^H$, so clearly $[\![\phi]\!]^H = N \supseteq V$.

If E is not simple, then Lemmas 8 and 9 tell us that $[\![E]\!]^H(v) - [\![r]\!]^H(v) \neq \emptyset$ for every $v \in V$. Since $[\![r]\!]^H = [\![p]\!]^H$, this means $H, v \nvDash \phi$ for $v \in V$, or, equivalently, $[\![\phi]\!]^G \cap V = \emptyset$. To see that, moreover, $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$, it remains to show that $G, v \vDash \phi$ iff $G', v \vDash \phi$ for all node names $v \notin V$. Clearly, $[\![p]\!]^G(v) = [\![p]\!]^{G'}(v) = \emptyset$. Now by Lemma 11, if E is safe, then also $[\![E]\!]^G(v) = [\![E]\!]^{G'}(v) = \emptyset$, so $G, v \vDash \phi$ and $G', v \vDash \phi$. On the other hand, if E is unsafe, then by the same Lemma $[\![E]\!]^G(v) = [\![E]\!]^{G'}(v) = \{v\} \neq \emptyset$, so $G, v \nvDash \phi$ and $G', v \nvDash \phi$, as desired.

Finally, assume ϕ is of the form $closed(R)$. If Σ contains a property name p not in R, then $[\![\phi]\!]^H \cap V = \emptyset$, since every node in H has an outgoing p-edge. Otherwise, i.e., if $\Sigma \subseteq R$, we have $[\![\phi]\!]^H \supseteq V$, since every node in H has only outgoing edges labeled by property names in Σ. To see that, moreover, $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$, it suffices to observe that trivially $H, v \vDash \phi$ for all node names $v \notin V$.

3.2 Closure

Without using *closed*, shapes cannot say anything about properties that they do not explicitly mention. We formalize this intuitive observation as follows. The proof is straightforward.

▶ **Lemma 12.** *Let Σ be a vocabulary, let E be a path expression over Σ, and let ϕ be a shape over Σ that does not use closed. Let G_1 and G_2 be graphs such that $[\![p]\!]^{G_1} = [\![p]\!]^{G_2}$ for every property name p in Σ. Then $[\![E]\!]^{G_1} = [\![E]\!]^{G_2}$ and $[\![\phi]\!]^{G_1} = [\![\phi]\!]^{G_2}$.*

Theorem 4 now follows readily for $X = closed$. Let F be a feature set without *closed*, let $\mathcal{S}$ be a shape schema in $\mathcal{L}(F)$, and let ϕ be the validation shape of $\mathcal{S}$. Let p be a property name not mentioned in $\mathcal{S}$, and different from r. Consider the graphs $G = \{(a, r, a), (a, p, a)\}$ and $G' = \{(a, r, a)\}$, so that G' belongs to Q_{closed} but G does not. By Lemma 12 we have $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$, showing that $\mathcal{S}$ does not define Q_{closed}.

▶ Remark 13. Lemma 12 fails completely in the presence of closure constraints. The simplest counterexample is to consider $\Sigma = \emptyset$ and the shape $closed(\emptyset)$. Trivially, any two graphs agree on the property names from Σ. However, $[\![closed(\emptyset)]\!]^G$, which equals the set of node names that do not have an outgoing edge in G (they may still have an incoming edge), obviously depends on the graph G.

3.3 Are target-based shape schemas enough?

Lemma 12 also allows us to clarify that, as far as expressive power is concerned, and in the absence of closure constraints, the restriction to target-based shape schemas is inconsequential.

▶ **Theorem 14.** *Every generalized shape schema that does not use closure constraints is equivalent to a target-based shape schema (that still does not use closure constraints).*

Proof. Let ϕ be the validation shape for shape schema $\mathcal{S}$, so that $G \vDash \mathcal{S}$ if and only if $[\![\phi]\!]^G$ is empty. Let C be the set of constants mentioned in ϕ.

Let us say that ϕ is *internal* if for every graph G and every node name v such that $G, v \vDash \phi$, we have $v \in N_G \cup C$. If ϕ is not internal, it can be shown that for every graph G and every node $v \notin N_G \cup C$, we have $G, v \vDash \phi$. Thus, if ϕ is not internal, $\mathcal{S}$ is unsatisfiable and is equivalent to the single target-based inclusion $\{c\} \subseteq \neg\top$, for an arbitrary constant c.

So now assume ϕ is internal. Define the target-based shape schema $\mathcal{T}$ consisting of the following inclusions:

- For each constant $c \in C$, the inclusion $\{c\} \subseteq \neg\phi$.
- For each property name mentioned in ϕ, the two inclusions $\exists p.\top \subseteq \neg\phi$ and $\exists p^-.\top \subseteq \neg\phi$.

We will show that $\mathcal{S}$ and $\mathcal{T}$ are equivalent. Let ψ be the validation shape for $\mathcal{T}$.

Let G be any graph, and let G' be the graph obtained from G by removing all triples involving property names not mentioned in ϕ. We reason as follows:

$$
\begin{aligned}
G \vDash \mathcal{S} &\Leftrightarrow [\![\phi]\!]^G = \emptyset \\
&\Leftrightarrow [\![\phi]\!]^{G'} = \emptyset && \text{by Lemma 12} \\
&\Leftrightarrow G' \vDash \mathcal{T} && \text{since } \phi \text{ is internal} \\
&\Leftrightarrow [\![\psi]\!]^{G'} = \emptyset \\
&\Leftrightarrow [\![\psi]\!]^G = \emptyset && \text{by Lemma 12} \\
&\Leftrightarrow G \vDash \mathcal{T}
\end{aligned}
$$

◀

▶ Remark 15. Note that we do not need class-based targets in the proof, so such targets are redundant on the left-hand sides of inclusions. This can also be seen directly: any inclusion

$$\exists \mathsf{type}/\mathsf{subclass}^*.\{c\} \subseteq \phi$$

with a class-based target is equivalent to the following inclusion with a subjects-of target:

$$\exists \mathsf{type}.\top \subseteq \neg\exists \mathsf{type}/\mathsf{subclass}^*.\{c\} \vee \phi$$

▶ Remark 16. Theorem 14 fails in the presence of closure constraints. For example, the inclusion $\neg closed(\emptyset) \subseteq \exists r.\top$ defines the class of graphs where every node with an outgoing edge has an outgoing r-edge. Suppose this inclusion would be equivalent to a target-based shape schema $\mathcal{S}$, and let R be the set of all property names mentioned in the targets of $\mathcal{S}$. Let p be a property name not in R and distinct from r; let a be a node name not used as a constant in $\mathcal{S}$; and consider the graph $G = \{(a, p, a)\}$. This graph trivially satisfies $\mathcal{S}$, but violates the inclusion.

4 Extension to stratified recursion

Until now, we could do without shape names. We do need them, however, for recursive shape schemas. Such schemas allow shapes to be defined using recursive rules, much as in Datalog and logic programming. The rules have a shape name in the head; in the body they have a shape that can refer to the same or other shape names.

▶ **Example 17.** The following rule defines a shape, named s, recursively:

$$s \leftarrow \{c\} \vee (eq(p,q) \wedge \exists r.s).$$

A node x will satisfy s if there is a path of r-edges from x to the constant c, so that all nodes along the path satisfy $eq(p,q)$ (for two property names p and q).

Rules and programs

We need to make a few extensions to our formalism and the semantics.

- We assume an infinite supply S of *shape names*. Again for simplicity of notation only, we assume that S is disjoint from N and P.
- The syntax of shapes is extended so that *every shape name is a shape*.
- A vocabulary Σ is now a subset of $N \cup P \cup S$; an interpretation I now additionally assigns a subset $[\![s]\!]^I$ of Δ^I to every shape name s in Σ.

Noting the obvious parallels with the field of logic programming, we propose to use the following terminology from that field. A *rule* is of the form $s \leftarrow \phi$, where s is a shape name and ϕ is a shape. A *program* is a finite set of rules. The shape names appearing as heads of rules in a program are called the *intensional* shape names of that program.

The following definitions of the semantics of programs are similar to definitions well-known for Datalog. A program is *semipositive* if for every intensional shape name s, and every shape ϕ in the body of some rule, s occurs only positively in ϕ. Let $\mathcal{P}$ be a semipositive program over vocabulary Σ, with set of intensional shape names D. An interpretation J over $\Sigma \cup D$ is called a *model* of $\mathcal{P}$ if for every rule $s \leftarrow \phi$ of $\mathcal{P}$, the set $[\![\phi]\!]^J$ is a subset of $[\![s]\!]^J$. Given any interpretation I over $\Sigma - D$, there exists a unique minimal interpretation J that expands I to $\Sigma \cup D$ such that J is a model of $\mathcal{P}$. We call J the result of applying $\mathcal{P}$ to I, and denote J by $\mathcal{P}(I)$.

Stratified programs are essentially sequences of semipositive programs. Formally, a program $\mathcal{P}$ is called *stratified* if it can be partitioned into parts $\mathcal{P}_1, \ldots, \mathcal{P}_n$ called *strata*, such that **(i)** the strata have pairwise disjoint sets of intensional shape names; **(ii)** each stratum is semipositive; and **(iii)** the strata are ordered in such a way that when a shape name s occurs in the body of a rule in some stratum, s is not intensional in any later stratum.

Let $\mathcal{P}$ be a stratified program with n strata $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and let again I be an interpretation over a vocabulary without the intensional shape names. We define $\mathcal{P}(I)$, the result of applying $\mathcal{P}$ to I, to be the interpretation J_n, where $J_0 := I$ and $J_{k+1} := \mathcal{P}_{k+1}(J_k)$ for $0 \leq k < n$.

Stratified shape schemas

We are now ready to define a *stratified shape schema* again as a set of inclusions, but now paired with a stratified program. Formally, it is a pair $(\mathcal{P}, \mathcal{T})$, where:

- $\mathcal{P}$ is a program that is stratified, and where every shape name mentioned in the body of some rule is an intensional shape name in $\mathcal{P}$.
- $\mathcal{T}$ is a finite set of inclusion statements $\phi_1 \subseteq \phi_2$, where ϕ_1 and ϕ_2 mention only shape names that are intensional in $\mathcal{P}$.

Now we define a graph G to *conform* to $(\mathcal{P}, \mathcal{T})$ if $[\![\phi_1]\!]^{\mathcal{P}(G)}$ is a subset of $[\![\phi_2]\!]^{\mathcal{P}(G)}$, for every inclusion $\phi_1 \subseteq \phi_2$ in $\mathcal{T}$.

▶ Remark 18. The nonrecursive notion of shape schema, defined in Section 2, corresponds to the special case where $\mathcal{P}$ is the empty program.

Extending Theorem 4

Theorem 4 extends to stratified shape schemas. Indeed, consider a stratified shape schema $(\mathcal{P}, \mathcal{T})$. Shapes not mentioning any shape names are referred to as *elementary shapes*. We observe that for every intensional shape name s and every graph H, there exists an elementary shape ϕ such that $[\![s]\!]^{\mathcal{P}(H)} = [\![\phi]\!]^H$. Furthermore, ϕ uses the same constants, quantifiers, and path expressions as $\mathcal{P}$. For semipositive programs, this is shown using a fixpoint characterization of the minimal model; for stratified programs, this argument can then be applied repeatedly. The crux, however, is that graphs G and G' of Proposition 5 will have the same ϕ. Indeed, by that Proposition, the fixpoints of the different strata will be reached on G and on G' in the same stage. We effectively obtain an extension of Proposition 5, which establishes the theorem for features X other than *closed*.

Also for $X =$ *closed*, the reasoning, given after Lemma 12, extends in the same way to stratified shape schemas, since the graphs G and G' used there again yield exactly the same evaluation for all shapes that do not use *closed*.

Extending Theorem 14

Also Theorem 14 extends to stratified shape schemas. Thereto, Lemma 12 needs to be reproven in the presence of a stratified program $\mathcal{P}$ defining the intensional shape names. The extended Lemma 12 then states that $[\![\phi]\!]^{\mathcal{P}(G)} = [\![\phi]\!]^{\mathcal{P}(G')}$. The proof of Theorem 14 then goes through unchanged.

5 Concluding remarks

An obvious open question is whether our results extend further to nonstratified programs, depending on various semantics that have been proposed for Datalog with negation, notably well-founded or stable models [1, 20]. One must then deal with 3-valued models and, for stable models, choose whether the TBox should hold in every stable model (skeptical), or in at least one (credulous). For example, Andreşel et al. [2] adopt a credulous approach. In the same vein, even for stratified programs, one may consider *maximal* models instead of minimal ones, as suggested for ShEx [6].

Notably, Corman et al. [9] have already suggested that disjointness is redundant in a setting of recursive shape schemas with nonstratified negation. Their expression is not correct, however [17].[4]

A quirk in the design of SHACL is that it only allows equality and disjointness tests $eq(E_1, E_2)$ and $disj(E_1, E_2)$ where E_1 can be a general path expression, but E_2 needs to be a property name. It is open whether allowing "full" equality or disjointness tests, i.e., allowing a general path expression for E_2, would strictly increase expressive power. Furthermore,

[4] Their approach is to postulate two shape names s_1 and s_2 that can be assigned arbitrary sets of nodes, as long as the two sets form a partition of the domain. Then for one node x to satisfy the shape $disj(E, p)$, it is sufficient that $E(x)$ is a subset of s_1 and $p(x)$ of s_2. This condition is not necessary, however, as other nodes may require different partitions.

for $X = disj$, our proof of Proposition 5, and thus Theorem 4, no longer works under full equality tests. For X one of the other three features, however, the proof continues to work under full disjointness or equality tests.

Not included in our formal syntax, but allowed in SHACL, are shapes of the form $eq(id, p)$ and $disj(id, p)$, where id stands for the focus node. These are allowed in SHACL under "node shapes". In our formalism, $eq(id, p)$ is already expressible as $eq(p?, p) \wedge \geq_1 p.\top \wedge \leq_1 p.\top$. Furthermore, $disj(id, p)$ is expressible as $\neg eq(p?, p)$. Note, however, the use of ? in these expressions. Hence, when investigating the primitivity of ?, as we have done in this paper, it would be more proper to include $eq(id, p)$ and $disj(id, p)$ directly in the syntax. Whether ? remains primitive in this setting is left as an open problem.

Another open problem, also concerning the primitivity of ?, is to consider the more restrictive setting where, in each shape of the form $\geq_n E.\phi$, n equals 1 or 2. Indeed, the higher counts seem rarely used in practice. Since our query showing primitivity of ? involves counting to 4, it remains open whether ? still adds expressive power when counts are limited to 1 or 2. (We thank an anonymous reviewer for suggesting this problem.)

Finally, a general question surrounding SHACL, even standard nonrecursive SHACL, is to understand better in which sense (if at all) this language is actually better suited for expressing constraints on RDF graphs than, say, SPARQL ASK queries [8, 19, 10]. Certainly, the affinity with description logics makes it easy to carve out cases where higher reasoning tasks become decidable [13, 15]. It is also possible to show that nonrecursive SHACL is strictly weaker in expressive power than SPARQL. But does SHACL conformance checking really have a lower computational complexity? Can we think of novel query processing strategies that apply to SHACL but not easily to SPARQL? Are SHACL expressions typically shorter, or perhaps longer, than the equivalent SPARQL ASK expression? How do the expression complexities [21] compare?

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 M. Andreşel, J. Corman, M. Ortiz, J.L. Reutter, O. Savkovic, and M. Simkus. Stable model semantics for recursive SHACL. In Y. Huang, I. King, T.-Y. Liu, and M. van Steen, editors, *Proceedings WWW'20*, pages 1570–1580. ACM, 2020.

3 A.K. Aylamazyan, M.M. Gilula, A.P. Stolboushkin, and G.F. Schwartz. Reduction of the relational model with infinite domains to the case of finite domains. *Doklady Akademii Nauk SSSR*, 286(2):308–311, 1986. In Russian.

4 F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

5 B. Bogaerts, M. Jakubowski, and J. Van den Bussche. SHACL: A description logic in disguise. *CoRR*, abs/2108.06096, 2021. URL: `https://arxiv.org/abs/2108.06096`.

6 I. Boneva, J.E.L. Gayo, and E.G. Prud'hommeaux. Semantics and validation of shape schemas for RDF. In C. d'Amato, M. Fernandez, V. Tamma, et al., editors, *Proceedings 16th International Semantic Web Conference*, volume 10587 of *Lecture Notes in Computer Science*, pages 104–120. Springer, 2017.

7 D. Calvanese, G. De Giacomo, D. Nardi, and M. Lenzerini. Reasoning in expressive description logics. In F. Baader, D. Calvanese, D. McGuiness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, chapter 23. Cambridge University Press, 2003.

8 J. Corman, F. Florenzano, J.L. Reutter, and O. Savkovic. Validating SHACL constraints over a SPARQL endpoint. In C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, et al., editors, *Proceedings 18th International Semantic Web Conference*, volume 11778 of *Lecture Notes in Computer Science*, pages 145–163. Springer, 2019.

9 J. Corman, J.L. Reutter, and O. Savkovic. Semantics and validation of recursive SHACL. In D. Vrandecic et al., editors, *Proceedings 17th International Semantic Web Conference*, volume 11136 of *Lecture Notes in Computer Science*, pages 318–336. Springer, 2018. Extended version, technical report KRDB18-01, `https://www.inf.unibz.it/krdb/tech-reports/`.

10 B. De Meester, P. Heyvaert, et al. RDF graph validation using rule-based reasoning. *Semantic Web*, 12(1):117–142, 2021.

11 I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In P. Doherty, J. Mylopoulos, and C.A. Welty, editors, *Proceedings 10th International Conference on Principles of Knowledge Representation and Reasoning*, pages 57–67. AAAI Press, 2006.

12 R. Hull and J. Su. Domain independence and the relational calculus. *Acta Informatica*, 31:513–524, 1994.

13 M. Leinberger, P. Seifer, et al. Deciding SHACL shape containment through description logics reasoning. In J.Z. Pan et al., editors, *Proceedings 19th International Semantic Web Conference*, volume 12506 of *Lecture Notes in Computer Science*, pages 366–383. Springer, 2020.

14 OWL 2 Web ontology language: Structural specification and functional-style syntax. W3C Recommendation, December 2012.

15 P. Pareti, G. Konstantinidis, et al. SHACL satisfiability and containment. In J.Z. Pan et al., editors, *Proceedings 19th International Semantic Web Conference*, volume 12506 of *Lecture Notes in Computer Science*, pages 474–493. Springer, 2020.

16 RDF 1.1 primer. W3C Working Group Note, June 2014.

17 J. Reutter, January 2021. Personal communication.

18 Shapes constraint language (SHACL). W3C Recommendation, July 2017.

19 J. Tao et al. Integrity constraints in OWL. In *Proceedings 24th AAAI Conference on Artificial Intelligence*, pages 1443–1448, 2010.

20 M. Truszczynski. An introduction to the stable and well-founded semantics of logic programs. In M. Kifer and Y.A. Liu, editors, *Declarative Logic Programming: Theory, Systems, and Applications*, pages 121–177. ACM and Morgan & Claypool, 2018.

21 M. Vardi. The complexity of relational query languages. In *Proceedings 14th ACM Symposium on the Theory of Computing*, pages 137–146, 1982.

Robustness Against Read Committed for Transaction Templates with Functional Constraints

Brecht Vandevoort
UHasselt, Data Science Institute, ACSL, Diepenbeek, Belgium

Bas Ketsman
Vrije Universiteit Brussel, Belgium

Christoph Koch
École Polytechnique Fédérale de Lausanne, Switzerland

Frank Neven
UHasselt, Data Science Institute, ACSL, Diepenbeek, Belgium

Abstract

The popular isolation level Multiversion Read Committed (RC) trades some of the strong guarantees of serializability for increased transaction throughput. Sometimes, transaction workloads can be safely executed under RC obtaining serializability at the lower cost of RC. Such workloads are said to be robust against RC. Previous work has yielded a tractable procedure for deciding robustness against RC for workloads generated by transaction programs modeled as transaction templates. An important insight of that work is that, by more accurately modeling transaction programs, we are able to recognize larger sets of workloads as robust. In this work, we increase the modeling power of transaction templates by extending them with functional constraints, which are useful for capturing data dependencies like foreign keys. We show that the incorporation of functional constraints can identify more workloads as robust that otherwise would not be. Even though we establish that the robustness problem becomes undecidable in its most general form, we show that various restrictions on functional constraints lead to decidable and even tractable fragments that can be used to model and test for robustness against RC for realistic scenarios.

2012 ACM Subject Classification Information systems → Database transaction processing

Keywords and phrases concurrency control, robustness, complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.16

Related Version *Full Version*: `https://arxiv.org/abs/2201.05021`

Funding This work is funded by FWO-grant G019921N.

1 Introduction

Many database systems implement several isolation levels, allowing users to trade isolation guarantees for improved performance. The highest, serializability, projects the appearance of a complete absence of concurrency, and thus perfect isolation. Executing transactions concurrently under weaker isolation levels can introduce certain anomalies. Sometimes, a transactional workload can be executed at an isolation level lower than serializability without introducing any anomalies. This is a desirable scenario: a lower isolation level, usually implementable with a cheaper concurrency control algorithm, yields the stronger isolation guarantees of serializability for free. This formal property is called robustness [12, 7]: a set of transactions $\mathcal{T}$ is called *robust against a given isolation level* if every possible interleaving of the transactions in $\mathcal{T}$ that is allowed under the specified isolation level is serializable.

Robustness received quite a bit of attention in the literature. Most existing work focuses on Snapshot Isolation (SI) [2, 4, 12, 13] or higher isolation levels [5, 7, 8, 10]. It is particularly interesting to consider robustness against lower level isolation levels like multi-version Read

Committed (referred to as RC from now on). Indeed, RC is widely available, often the default in database systems (see, e.g., [4]), and is generally expected to have better throughput than stronger isolation levels.

In previous work [17], we provided a tractable decision procedure for robustness against RC for workloads generated by transaction programs modeled as transaction templates. The approach is centered on a novel characterization of robustness against RC in the spirit of [12, 14] that improves over the sufficient condition presented in [3], and on a formalization of transaction programs, called *transaction templates*, facilitating fine-grained reasoning for robustness against RC. Conceptually, transaction templates as introduced in [17] are functions with parameters, and can, for instance, be derived from stored procedures inside a database system (c.f. Figure 1 for an example). The abstraction generalizes transactions as usually studied in concurrency control research – sequences of read and write operations – by making the objects worked on variable, determined by input parameters. Such parameters are *typed* to add additional power to the analysis. They support *atomic updates* (that is, a read followed by a write of the same database object, to make a relative change to its value). Furthermore, database objects read and written are considered at the granularity of fields, rather than just entire tuples, decoupling conflicts further and allowing to recognize additional cases that would not be recognizable as robust on the tuple level.

An important insight obtained from [17] is that more accurate modeling of the workload allows to recognize larger sets of transaction programs as robust. Processing workloads under RC increases the throughput of the transactional database system compared to when executing the workload under SI or serializable SI, so larger robust sets mean better performance of the database system. In this work, we increase the modeling power of transaction templates by extending them with *functional constraints*, which are useful for capturing data dependencies like foreign keys (inclusion dependencies). This appears to be a sweet spot for strengthening modelling power – as we show in this paper, it allows us to remain with abstractions that have been well established within database theory, without having to move to general program analysis, and it pushes the robustness frontier on popular transaction processing benchmarks. Generally speaking, workloads can profit more from richer modelling the larger and more complex they get, so the fact that adding functional constraints yields larger robust sets already on these simple benchmarks suggests that these techniques are practically useful. Our contributions can be summarized as follows:

- We argue in Section 2 through the SmallBank and TPC-C benchmarks that the incorporation of functional constraints can identify more workloads as robust that otherwise would not be, and that they reduce the extent to which changes need to be made to workloads to make them robust against RC.
- In Section 4, we establish that robustness in its most general form becomes undecidable. The proof is a reduction from PCP and relies on cyclic dependencies between functions allowing to connect data values through an unbounded application of functions.
- We consider a fragment in Section 5 that only allows a very limited form of cyclic dependencies between functions and assumes additional constraints on templates that, together, imply that functions behave as bijections. Robustness against RC can be decided in NLOGSPACE and this fragment is general enough to model the SmallBank benchmark.
- In Section 6, we obtain an EXPSPACE decision procedure when the schema graph is acyclic (so, no cyclic dependencies between functions). Even for small input sizes, such a result is not practical. We provide various restrictions that lower the complexity to PSPACE and EXPTIME, and which allow to model the TPC-C benchmark as discussed. Notice that, for robustness testing, an exponential time decision procedure is considered to be practical as the size of the input is small and robustness is a static property that can be tested offline.

GoPremium:

$\mathtt{U}[\mathtt{X} : \text{Account}\{\text{N, C}\}\{\text{I}\}]$
$\mathtt{R}[\mathtt{Y} : \text{Savings}\{\text{C, I}\}]$
$\mathtt{U}[\mathtt{Y} : \text{Savings}\{\text{C}\}\{\text{I}\}]$
$\mathtt{Y} = f_{A\to S}(\mathtt{X}),\ \mathtt{X} = f_{S\to A}(\mathtt{Y})$

Figure 1 Transaction template.

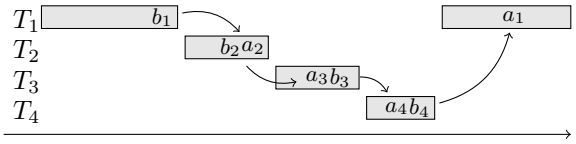


Figure 2 Multiversion split schedule.

These contributions should be contrasted with our earlier work [17], where we focused on a characterization for robustness against RC for basic transaction templates without functional constraints and performed an experimental study to show how the robustness property can improve transaction throughput.

Due to space constraints, proofs as well as a more complete description of the SmallBank and TPC-C benchmarks are moved to the online available full version of this paper [18].

2 Application

We present a small extension of the SmallBank benchmark [2] to exemplify the modeling power of transaction templates and discuss how the addition of functional constraints can detect larger sets of transaction templates to be robust. Finally, we discuss in the context of the TPC-C benchmark how the incorporation of functional constraints requires less changes to templates in making them robust. A full description of these benchmarks can be found in the online available full version [18].

The SmallBank schema consists of three tables: Account(Name, CustomerID, IsPremium), Savings(CustomerID, Balance, InterestRate), and Checking(CustomerID, Balance). Underlined attributes are primary keys. The Account table associates customer names with IDs and keeps track of the premium status (Boolean); CustomerID is a `UNIQUE` attribute. The other tables contain the balance (numeric value) of the savings and checking accounts of customers identified by their ID. Account (CustomerID) is a foreign key referencing both the columns Savings (CustomerID) and Checking (CustomerID). The interest rate on a savings account is based on a number of parameters, including the account status (premium or not). The application code can interact with the database through a fixed number of transaction programs: Balance, TransactSavings, Amalgamate, WriteCheck, DepositChecking, and GoPremium. We only discuss GoPremium(N), given in Figure 1, which converts the account of the customer with name N to a premium account and updates the interest rate of the corresponding savings account.

In short, a transaction template is a sequence of read (`R`), write (`W`) and update (`U`) statements over typed variables (`X`, `Y`, ...) with additional equality and disequality constraints. For instance, $\mathtt{R}[\mathtt{Y} : \text{Savings}\{\mathtt{C}, \mathtt{I}\}]$ indicates that a read operation is performed to a tuple in relation Savings on the attributes CustomerID and InterestRate. We abbreviate the names of attributes by their first letter to save space. The set $\{C, I\}$ is the read set. Write operations have an associated write set while update operations contain a read set followed by a write set: e.g., $\mathtt{U}[\mathtt{X} : \text{Account}\{\mathtt{N}, \mathtt{C}\}\{\mathtt{I}\}]$ first reads the Name and CustomerID of tuple `X` and then writes to the attribute InterestRate. To capture the dependencies between tuples induced by the foreign keys, we use two unary functions: $f_{A\to S}$ maps a tuple of type Account to a tuple of type Savings, while $f_{A\to C}$ maps a tuple of type Account to a tuple of type Checking. As Account(CustomerID) is `UNIQUE`, every savings and checking accounts is associated to

a unique Account tuple. This is modelled through the functions $f_{C\to A}$ and $f_{S\to A}$ with an analogous interpretation. Notice that the equality constraints for GoPremium imply that these functions are bijections and each others inverses.

A transaction T over a database $\mathbf{D}$ is an *instantiation* of a transaction template τ if there is a variable mapping μ from the variables in τ to tuples in $\mathbf{D}$ that satisfies all the constraints in τ such that with $\mu(\tau) = T$. For instance, consider a database $\mathbf{D}$ with tuples $\mathtt{a}_1, \mathtt{a}_2, \ldots$ of type Account, $\mathtt{s}_1, \mathtt{s}_2, \ldots$ of type Savings, and $\mathtt{c}_1, \mathtt{c}_2, \ldots$ of type Checking with $f^{\mathbf{D}}_{A\to S}(\mathtt{a}_i) = \mathtt{s}_i$, $f^{\mathbf{D}}_{A\to C}(\mathtt{a}_i) = \mathtt{c}_i$, $f^{\mathbf{D}}_{S\to A}(\mathtt{s}_i) = \mathtt{a}_i$, $f^{\mathbf{D}}_{C\to A}(\mathtt{c}_i) = \mathtt{a}_i$ for each i. Then, for $\mu_1 = \{\mathtt{X} \to \mathtt{a}_1, \mathtt{Y} \to \mathtt{s}_1\}$, $\mu_1(\text{GoPremium}) = \mathtt{U}[\mathtt{a}_1]\mathtt{R}[\mathtt{s}_1]\mathtt{U}[\mathtt{s}_1]$ is an instantiation of GoPremium whereas $\mu_2(\text{GoPremium})$ with $\mu_2 = \{\mathtt{X} \to \mathtt{a}_1, \mathtt{Y} \to \mathtt{s}_2\}$ is not as the functional constraint $\mathtt{Y} = f_{A\to S}(\mathtt{X})$ is not satisfied. Indeed, $\mu_2(\mathtt{Y}) = \mathtt{s}_2 \neq \mathtt{s}_1 = f^{\mathbf{D}}_{A\to S}(\mathtt{a}_1) = f^{\mathbf{D}}_{A\to S}(\mu_2(\mathtt{X}))$. We then say that a set of transactions is *consistent* with a set of templates if every transaction is an instantiation of a transaction template.

Functional constraints do not replace the more usual data consistency constraints like key constraints, functional dependencies or denial constraints, ... The latter are intended to verify data consistency, whereas the former are intended to verify whether a set of transactions instantiated from templates are indeed consistent with these templates. The abstraction of functional constraints provides a straightforward mechanism to capture dependencies between tuples implied by e.g. foreign key constraints. Consider for example variables `X` and `Y` in GoPremium. Rather than specifying that the value of the attribute CustomerID in the tuple assigned to `X` should agree with the value of the attribute CustomerID in the tuple assigned to `Y` and combining this information with the defined foreign key from Account to Savings to conclude that two instantiations of GoPremium that agree on the tuple assigned to `X` should also agree on the tuple assigned to `Y`, the functional constraint $\mathtt{Y} = f_{A\to S}(\mathtt{X})$ expresses this dependency more directly. An additional benefit of our abstraction is that this approach is not limited to dependencies implied by foreign keys. For the SmallBank benchmark, for example, we can infer from the fact that Account(CustomerID) is `UNIQUE` that each checking and savings account is associated to exactly one Account tuple, even though no foreign key from respectively Checking and Savings to Account is defined in the schema.

Our previous work [17], which did not consider functional constraints, has shown that {Am,DC,TS}, {Bal,DC}, and {Bal,TS} are maximal robust sets of transaction templates. This means that for any database, for any set of transactions $\mathcal{T}$ that is consistent with one of the three mentioned sets, any possible interleaving of the transactions in $\mathcal{T}$ that is allowed under RC is *always* serializable! Using the results from Section 5, it follows that when functional constraints are taken into account GoPremium can be added to each of these sets as well: {Am,DC,GP,TS}, {Bal,DC,GP}, {Bal,TS,GP} are maximal robust sets.

We argue that incorporating functional constraints is crucial. Indeed, without functional constraints it's easy to show that even the set {GoPremium} is not robust. Consider the schedule over two instantiations T_1 and T_2 of GoPremium, where we use the mappings μ_1 and μ_2 as defined above for respectively T_1 and T_2 (we show the read and write sets to facilitate the discussion):

$$
\begin{array}{ll}
T_1: & \mathtt{U}_1[\mathtt{a}_1\{\text{N,C}\}\{\text{I}\}]\,\mathtt{R}_1[\mathtt{s}_1\{\text{C,I}\}] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{U}_1[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]\,\mathtt{C}_1 \\
T_2: & \qquad\qquad\qquad\qquad \mathtt{U}_2[\mathtt{a}_2\{\text{N,C}\}\{\text{I}\}]\,\mathtt{R}_2[\mathtt{s}_1\{\text{C,I}\}]\,\mathtt{U}_2[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]\,\mathtt{C}_2
\end{array}
$$

The above schedule is allowed under RC as there is no dirty write, but it is not conflict serializable. Indeed, there is a rw-conflict between $\mathtt{R}_1[\mathtt{s}_1\{\text{C,I}\}]$ and $\mathtt{U}_2[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]$ as the former reads the attribute I that is written to by the latter, which implies that T_1 should occur before

Delivery:

$\mathtt{U}[\mathtt{S} : \text{Order}\{\text{W, D, O}\}\{\text{Sta}\}]$
$\mathtt{U}[\mathtt{V}_1 : \text{OrderLine}\{\text{W, D, O, OL, Del}\}\{\text{Del}\}]$
$\mathtt{U}[\mathtt{V}_2 : \text{OrderLine}\{\text{W, D, O, OL, Del}\}\{\text{Del}\}]$
$\mathtt{U}[\mathtt{Z} : \text{Customer}\{\text{W, D, C, Bal}\}\{\text{Bal}\}]$
$\mathtt{Z} = f_{O\to C}(S),\ S = f_{L\to O}(\mathtt{V}_1),\ S = f_{L\to O}(\mathtt{V}_2)$

OrderStatus:

$\mathtt{R}[\mathtt{Z} : \text{Customer}\{\text{W, D, C, Inf, Bal}\}]$
$\mathtt{R}[\mathtt{S} : \text{Order}\{\text{W, D, O, C, Sta}\}]$
$\mathtt{R}[\mathtt{V}_1 : \text{OrderLine}\{\text{W, D, O, OL, I, Del, Qua}\}]$
$\mathtt{R}[\mathtt{V}_2 : \text{OrderLine}\{\text{W, D, O, OL, I, Del, Qua}\}]$
$\mathtt{Z} = f_{O\to C}(S),\ S = f_{L\to O}(\mathtt{V}_1),\ S = f_{L\to O}(\mathtt{V}_2)$

Figure 3 Transaction templates Delivery and OrderStatus of the TPC-C benchmark.

T_2 in an equivalent serial schedule. But, there is a ww-conflict between $\mathtt{U}_2[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]$ and $\mathtt{U}_1[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]$ as both write to the common attribute I implying that T_2 should occur before T_1 in an equivalent serial schedule. Consequently, the schedule is not serializable. However, taking functional constraints into account, $\{T_1, T_2\}$ is not consistent with $\{\text{GoPremium}\}$ as $\mu_2(\mathtt{Y}) = \mathtt{s}_1 \neq \mathtt{s}_2 = f_{A\to S}(a_2) = f_{A\to S}(\mu_2(\mathtt{X}))$ implying that the above schedule is *not* a counter example for robustness.

Incorporating functional constraints for TPC-C can not identify larger sets of templates to be robust. However, when a set of transaction templates $\mathcal{P}$ is not robust against RC, an equivalent set of templates $\mathcal{P}'$ can be constructed from $\mathcal{P}$ by *promoting* certain R-operations to U-operations [17]. By incorporating functional constraints it can be shown that fewer R-operations need to be promoted leading to an increase in throughput as R-operations do not take locks whereas U-operations do. Consider for example the subset $\mathcal{P} = \{\text{Delivery}, \text{OrderStatus}\}$ of the TPC-C benchmark, given in Figure 3, where functional constraints are added to express the fact that a tuple of type OrderLine implies the tuple of type Order, which in turn implies the tuple of type Customer. This set $\mathcal{P}$ is not robust against RC, but robustness can be achieved by promoting the R-operation over Customer in OrderStatus to a U-operation. However, without functional constraints, this single promoted operation no longer guarantees robustness, as witnessed by the following schedule:

$$\begin{array}{lll} T_1(\text{Orderstatus}): & \mathtt{U}_1[\mathtt{c}]\,\mathtt{R}_1[\mathtt{a}] & \mathtt{R}_1[\mathtt{b}_1]\,\mathtt{R}_1[\mathtt{b}_2]\,\mathtt{C}_1 \\ T_2(\text{Delivery}): & \mathtt{U}_2[\mathtt{a}]\,\mathtt{U}_2[\mathtt{b}_1]\,\mathtt{U}_2[\mathtt{b}_2]\,\mathtt{U}_2[\mathtt{c}']\,\mathtt{C}_2 & \end{array}$$

Notice in particular how this schedule implicitly assumes in T_2 that Order $\mathtt{a}$ belongs to Customer $\mathtt{c}'$ instead of Customer $\mathtt{c}$ to avoid a dirty write on $\mathtt{c}$. Without functional constraints, $\mathcal{P}$ is only robust against RC if *all* R-operations in OrderStatus are promoted to U-operations.

3 Definitions

We recall the necessary definitions from [17] and extend them with functional constraints.

3.1 Databases

A *relational schema* is a pair $(\mathsf{Rels}, \mathsf{Funcs})$ where Rels is a set of relation names and Funcs is a set of function names. A finite set of attribute names $\text{Attr}(R)$ is associated to every relation $R \in \mathsf{Rels}$. Relations will be instantiated by abstract objects that serve as an abstraction of relational tuples. To this end, for every relation $R \in \mathsf{Rels}$, we fix an infinite set of tuples $\mathbf{Tuples}_R$. Furthermore, we assume that $\mathbf{Tuples}_R \cap \mathbf{Tuples}_S = \emptyset$ for all $R, S \in \mathsf{Rels}$ with $R \neq S$. We then denote by $\mathbf{Tuples}$ the set $\bigcup_{R\in\mathsf{Rels}} \mathbf{Tuples}_R$ of all possible tuples. Notice that, by definition, for every $\mathtt{t} \in \mathbf{Tuples}$ there is a unique relation $R \in \mathsf{Rels}$ such that $\mathtt{t} \in \mathbf{Tuples}_R$. In that case, we say that $\mathtt{t}$ is of *type* R and denote the latter by $\text{type}(\mathtt{t}) = R$. Each function name $f \in \mathsf{Funcs}$ has a domain $dom(f) \in \mathsf{Rels}$ and a range $range(f) \in \mathsf{Rels}$.

Functions are used to encode relationships between tuples like for instance those implied by foreign-keys constraints. For instance, in the SmallBank example $\mathsf{Funcs} = \{f_{A\to S}, f_{A\to C}\}$, $dom(f_{A\to S}) = dom(f_{A\to C}) = A$, $range(f_{A\to S}) = S$, and $range(f_{A\to C}) = C$. A *database* $\mathbf{D}$ over schema $(\mathsf{Rels}, \mathsf{Funcs})$ assigns to every relation name $R \in \mathsf{Rels}$ a finite set $R^{\mathbf{D}} \subset \mathbf{Tuples}_R$ and to every function name $f \in \mathsf{Funcs}$ a function $f^{\mathbf{D}}$ from $dom(f)^{\mathbf{D}}$ to $range(f)^{\mathbf{D}}$.

3.2 Transactions and Schedules

For a tuple $\mathtt{t} \in \mathbf{Tuples}$, we distinguish three operations $\mathtt{R}[\mathtt{t}]$, $\mathtt{W}[\mathtt{t}]$, and $\mathtt{U}[\mathtt{t}]$ on $\mathtt{t}$, denoting that tuple $\mathtt{t}$ is read, written, or updated, respectively. We say that the operation is on the tuple $\mathtt{t}$. The operation $\mathtt{U}[\mathtt{t}]$ is an atomic update and should be viewed as an atomic sequence of a read of $\mathtt{t}$ followed by a write to $\mathtt{t}$. We will use the following terminology: a *read operation* is an $\mathtt{R}[\mathtt{t}]$ or a $\mathtt{U}[\mathtt{t}]$, and a *write operation* is a $\mathtt{W}[\mathtt{t}]$ or a $\mathtt{U}[\mathtt{t}]$. Furthermore, an $\mathtt{R}$-operation is an $\mathtt{R}[\mathtt{t}]$, a $\mathtt{W}$-operation is a $\mathtt{W}[\mathtt{t}]$, and a $\mathtt{U}$-operation is a $\mathtt{U}[\mathtt{t}]$. We also assume a special *commit* operation denoted $\mathtt{C}$. To every operation o on a tuple of type R, we associate the set of attributes $\mathrm{ReadSet}(o) \subseteq \mathrm{Attr}(R)$ and $\mathrm{WriteSet}(o) \subseteq \mathrm{Attr}(R)$ containing, respectively, the set of attributes that o reads from and writes to. When o is a $\mathtt{R}$-operation then $\mathrm{WriteSet}(o) = \emptyset$. Similarly, when o is a $\mathtt{W}$-operation then $\mathrm{ReadSet}(o) = \emptyset$.

A *transaction* T is a sequence of read and write operations followed by a commit. We assume that a transactions starts when its first operation is executed, but no earlier. Formally, we model a transaction as a linear order $(T, \leq_T)$, where T is the set of (read, write and commit) operations occurring in the transaction and $\leq_T$ encodes the ordering of the operations. As usual, we use $<_T$ to denote the strict ordering.

When considering a set $\mathcal{T}$ of transactions, we assume that every transaction in the set has a unique id i and write T_i to make this id explicit. Similarly, to distinguish the operations of different transactions, we add this id as a subscript to the operation. That is, we write $\mathtt{W}_i[\mathtt{t}]$, $\mathtt{R}_i[\mathtt{t}]$, and $\mathtt{U}_i[\mathtt{t}]$ to denote a $\mathtt{W}[\mathtt{t}]$, $\mathtt{R}[\mathtt{t}]$, and $\mathtt{U}[\mathtt{t}]$ occurring in transaction T_i; similarly $\mathtt{C}_i$ denotes the commit operation in transaction T_i. This convention is consistent with the literature (see, *e.g.* [6, 12]). To avoid ambiguity of notation, we assume that a transaction performs at most one write, one read, and one update per tuple. The latter is a common assumption (see, *e.g.* [12]). All our results carry over to the more general setting in which multiple writes and reads per tuple are allowed.

A *(multiversion) schedule* s over a set $\mathcal{T}$ of transactions is a tuple $(O_s, \leq_s, \ll_s, v_s)$ where O_s is the set containing all operations of transactions in $\mathcal{T}$ as well as a special operation op_0 conceptually writing the initial versions of all existing tuples, $\leq_s$ encodes the ordering of these operations, $\ll_s$ is a *version order* providing for each tuple $\mathtt{t}$ a total order over all write operations on $\mathtt{t}$ occurring in s, and v_s is a *version function* mapping each read operation a in s to either op_0 or to a write[1] operation different from a in s. We require that $op_0 \leq_s a$ for every operation $a \in O_s$, $op_0 \ll_s a$ for every write operation $a \in O_s$, and that $a <_T b$ implies $a <_s b$ for every $T \in \mathcal{T}$ and every $a, b \in T$.[2] We furthermore require that for every read operation a, $v_s(a) <_s a$ and, if $v_s(a) \neq op_0$, then the operation $v_s(a)$ is on the same tuple as a. Intuitively, op_0 indicates the start of the schedule, the order of operations in s is consistent with the order of operations in every transaction $T \in \mathcal{T}$, and the version function maps each read operation a to the operation that wrote the version observed by a. If $v_s(a)$ is op_0, then a observes the initial version of this tuple. The version order $\ll_s$ represents the

[1] Recall that a write operation is either a $\mathtt{W}[\mathtt{x}]$ or a $\mathtt{U}[\mathtt{x}]$.
[2] Recall that $<_T$ denotes the order of operations in transaction T.

order in which different versions of a tuple are installed in the database. For a pair of write operations on the same tuple, this version order does not necessarily coincide with $\leq_s$. For example, under RC the version order is based on the commit order instead.

We say that a schedule s is a *single version schedule* if $\ll_s$ coincides with $\leq_s$ and every read operation always reads the last written version of the tuple. Formally, for each pair of write operations a and b on the same tuple, $a \ll_s b$ iff $a <_s b$, and for every read operation a there is no write operation c on the same tuple as a with $v_s(a) <_s c <_s a$. A single version schedule over a set of transactions $\mathcal{T}$ is *single version serial* if its transactions are not interleaved with operations from other transactions. That is, for every $a, b, c \in O_s$ with $a <_s b <_s c$ and $a, c \in T$ implies $b \in T$ for every $T \in \mathcal{T}$.

The absence of aborts in our definition of schedule is consistent with the common assumption [12, 7] that an underlying recovery mechanism will rollback aborted transactions. We only consider isolation levels that only read committed versions. Therefore there will never be cascading aborts.

3.3 Conflict Serializability

Let a_j and b_i be two operations on the same tuple from different transactions T_j and T_i in a set of transactions $\mathcal{T}$. We then say that a_j is *conflicting* with b_i if:

- *(ww-conflict)* $\text{WriteSet}(a_j) \cap \text{WriteSet}(b_i) \neq \emptyset$; or,
- *(wr-conflict)* $\text{WriteSet}(a_j) \cap \text{ReadSet}(b_i) \neq \emptyset$; or,
- *(rw-conflict)* $\text{ReadSet}(a_j) \cap \text{WriteSet}(b_i) \neq \emptyset$.

In this case, we also say that a_j and b_i are conflicting operations. Furthermore, commit operations and the special operation op_0 never conflict with any other operation. When a_j and b_i are conflicting operations in $\mathcal{T}$, we say that a_j *depends on* b_i in a schedule s over $\mathcal{T}$, denoted $b_i \rightarrow_s a_j$ if:[3]

- *(ww-dependency)* b_i is ww-conflicting with a_j and $b_i \ll_s a_j$; or,
- *(wr-dependency)* b_i is wr-conflicting with a_j and $b_i = v_s(a_j)$ or $b_i \ll_s v_s(a_j)$; or,
- *(rw-antidependency)* b_i is rw-conflicting with a_j and $v_s(b_i) \ll_s a_j$.

Intuitively, a ww-dependency from b_i to a_j implies that a_j writes a version of a tuple that is installed after the version written by b_i. A wr-dependency from b_i to a_j implies that b_i either writes the version observed by a_j, or it writes a version that is installed before the version observed by a_j. A rw-antidependency from b_i to a_j implies that b_i observes a version installed before the version written by a_j.

Two schedules s and s' are *conflict equivalent* if they are over the same set $\mathcal{T}$ of transactions and for every pair of conflicting operations a_j and b_i, $b_i \rightarrow_s a_j$ iff $b_i \rightarrow_{s'} a_j$.

▶ **Definition 1.** *A schedule s is* conflict serializable *if it is conflict equivalent to a single version serial schedule.*

A *conflict graph* $CG(s)$ for schedule s over a set of transactions $\mathcal{T}$ is the graph whose nodes are the transactions in $\mathcal{T}$ and where there is an edge from T_i to T_j if T_i has an operation b_i that conflicts with an operation a_j in T_j and $b_i \rightarrow_s a_j$.

▶ **Theorem 2** ([15])**.** *A schedule s is conflict serializable iff the conflict graph for s is acyclic.*

[3] Throughout the paper, we adopt the following convention: a b operation can be understood as a "before" while an a can be interpreted as an "after".

3.4 Multiversion Read Committed

Let s be a schedule for a set $\mathcal{T}$ of transactions. Then, s *exhibits a dirty write* iff there are two ww-conflicting operations a_j and b_i in s on the same tuple $\mathtt{t}$ with $a_j \in T_j$, $b_i \in T_i$ and $T_j \neq T_i$ such that $b_i <_s a_j <_s \mathtt{C}_i$. That is, transaction T_j writes to an attribute of a tuple that has been modified earlier by T_i, but T_i has not yet issued a commit.

For a schedule s, the version order $\ll_s$ corresponds to the commit order in s if for every pair of write operations $a_j \in T_j$ and $b_i \in T_i$, $b_i \ll_s a_j$ iff $\mathtt{C}_i <_s a_j$. We say that a schedule s is *read-last-committed (RLC)* if $\ll_s$ corresponds to the commit order and for every read operation a_j in s on some tuple $\mathtt{t}$ the following holds:

- $v_s(a_j) = op_0$ or $\mathtt{C}_i <_s a_j$ with $v_s(a_j) \in T_i$; and
- there is no write[4] operation $c_k \in T_k$ on $\mathtt{t}$ with $\mathtt{C}_k <_s a_j$ and $v_s(a_j) \ll_s c_k$.

So, a_j observes the most recent version of $\mathtt{t}$ (according to the order of commits) that is committed before a_j. Note in particular that a schedule cannot exhibit dirty reads, defined in the traditional way [6], if it is read-last-committed.

▶ **Definition 3.** *A schedule is* allowed under isolation level *read committed (RC) if it is read-last-committed and does not exhibit dirty writes.*

3.5 Transaction Templates

Transaction templates are transactions where operations are defined over typed variables together with functional constraints on these variables. Types of variables are relation names in Rels and indicate that variables can only be instantiated by tuples from the respective type. We fix an infinite set of variables $\mathbf{Var}$ that is disjoint from $\mathbf{Tuples}$. Every variable $\mathtt{X} \in \mathbf{Var}$ has an associated relation name in Rels as type that we denote by $\mathrm{type}(\mathtt{X})$. For an operation o_i in a template, $var(o_i)$ denotes the variable in o_i. An *equality constraint* is an expression of the form $\mathtt{X} = f(\mathtt{Y})$ where $\mathtt{X}, \mathtt{Y} \in \mathbf{Var}$, $dom(f) = \mathrm{type}(\mathtt{Y})$ and $range(f) = \mathrm{type}(\mathtt{X})$. A *disequality constraint* is an expression of the form $\mathtt{X} \neq \mathtt{Y}$ where $\mathrm{type}(\mathtt{X}) = \mathrm{type}(\mathtt{Y})$.

▶ **Definition 4.** *A* transaction template *is a transaction τ over $\mathbf{Var}$ together with a set $\Gamma(\tau)$ of equality and disequality constraints. In addition, for every operation o in τ over a variable $\mathtt{X}$, $ReadSet(o) \subseteq Attr(type(\mathtt{X}))$ and $WriteSet(o) \subseteq Attr(type(\mathtt{X}))$.*

Recall that we denote variables by capital letters $\mathtt{X}, \mathtt{Y}, \mathtt{Z}$ and tuples by small letters $\mathtt{t}, \mathtt{v}$. A variable assignment μ is a mapping from $\mathbf{Var}$ to $\mathbf{Tuples}$ such that $\mu(\mathtt{X}) \in \mathbf{Tuples}_{\mathrm{type}(\mathtt{X})}$. Furthermore, μ *satisfies* a constraint $\mathtt{X} = f(\mathtt{Y})$ (resp., $\mathtt{X} \neq \mathtt{Y}$) over a database $\mathbf{D}$ when $\mu(\mathtt{X}) = f^{\mathbf{D}}(\mu(\mathtt{Y}))$ (resp., $\mu(\mathtt{X}) \neq \mu(\mathtt{Y})$). A variable assignment μ for a transaction template τ is *admissible* for $\mathbf{D}$ if it satisfies all constraints in $\Gamma(\tau)$ over $\mathbf{D}$. By $\mu(\tau)$, we denote the transaction obtained by replacing each variable $\mathtt{X}$ in τ with $\mu(\mathtt{X})$.

A set of transactions $\mathcal{T}$ is *consistent* with a set of transaction templates $\mathcal{P}$ and database $\mathbf{D}$, if for every transaction T in $\mathcal{T}$ there is a transaction template $\tau \in \mathcal{P}$ and a variable mapping μ_T that is admissible for $\mathbf{D}$ such that $\mu_T(\tau) = T$.

3.6 Robustness

We define the robustness property [7] (also called *acceptability* in [12, 13]), which guarantees serializability for all schedules of a given set of transactions for a given isolation level.

[4] Recall that a write operation is either a $\mathtt{W}$ or a $\mathtt{U}$-operation.

▶ **Definition 5** (Transaction Robustness). *A set $\mathcal{T}$ of transactions is* robust *against RC if every schedule for $\mathcal{T}$ that is allowed under RC is conflict serializable.*

In the next definition, we represent conflicting operations from transactions in a set $\mathcal{T}$ as quadruples (T_i, b_i, a_j, T_j) with b_i and a_j conflicting operations, and T_i and T_j their respective transactions in $\mathcal{T}$. We call these quadruples *conflicting quadruples* for $\mathcal{T}$. Further, for an operation $b \in T$, we denote by $\mathsf{prefix}_b(T)$ the restriction of T to all operations that are before or equal to b according to $\leq_T$. Similarly, we denote by $\mathsf{postfix}_b(T)$ the restriction of T to all operations that are strictly after b according to $\leq_T$. Throughout the paper, we interchangeably consider transactions both as linear orders as well as sequences. Therefore, T is then equal to the sequence $\mathsf{prefix}_b(T)$ followed by $\mathsf{postfix}_b(T)$ which we denote by $\mathsf{prefix}_b(T) \cdot \mathsf{postfix}_b(T)$ for every $b \in T$.

▶ **Definition 6** (Multiversion split schedule). *Let $\mathcal{T}$ be a set of transactions and $C = (T_1, b_1, a_2, T_2), (T_2, b_2, a_3, T_3), \ldots, (T_m, b_m, a_1, T_1)$ a sequence of conflicting quadruples for $\mathcal{T}$ such that each transaction in $\mathcal{T}$ occurs in at most two different quadruples. A* multiversion split schedule *for $\mathcal{T}$ based on C is a multiversion schedule that has the following form:*

$$\mathsf{prefix}_{b_1}(T_1) \cdot T_2 \cdot \ldots \cdot T_m \cdot \mathsf{postfix}_{b_1}(T_1) \cdot T_{m+1} \cdot \ldots \cdot T_n,$$

where

1. *there is no write operation in $\mathsf{prefix}_{b_1}(T_1)$ ww-conflicting with a write operation in any of the transactions $T_2, \ldots, T_m$;*
2. *$b_1 <_{T_1} a_1$ or b_m is rw-conflicting with a_1; and,*
3. *b_1 is rw-conflicting with a_2.*

Furthermore, $T_{m+1}, \ldots, T_n$ are the remaining transactions in $\mathcal{T}$ (those not mentioned in C) in an arbitrary order.

Figure 2 depicts a schematic multiversion split schedule. The name stems from the fact that the schedule is obtained by splitting one transaction in two (T_1 at operation b_1 in Figure 2) and placing all other transactions in C in between. The figure does not display the trailing transactions $T_{m+1}, T_{m+2}, \ldots$ and assumes $b_1 <_{T_1} a_1$.

The following theorem characterizes non-robustness in terms of the existence of a multiversion split schedule.

▶ **Theorem 7** ([17]). *For a set of transactions $\mathcal{T}$, the following are equivalent:*

1. *$\mathcal{T}$ is not robust against RC;*
2. *there is a multiversion split schedule s for $\mathcal{T}$ based on some C.*

Let $\mathcal{P}$ be a set of transaction templates and $\mathbf{D}$ be a database. Then, $\mathcal{P}$ is *robust against RC over $\boldsymbol{D}$* if for every set of transactions $\mathcal{T}$ that is consistent with $\mathcal{P}$ and $\mathbf{D}$, it holds that $\mathcal{T}$ is robust against RC.

▶ **Definition 8** (Template Robustness). *A set of transaction templates $\mathcal{P}$ is* robust *against RC if $\mathcal{P}$ is robust against RC for every database $\boldsymbol{D}$.*

We say that a transaction template (τ, Γ) is a *variable transaction template* when $\Gamma = \emptyset$ and an *equality transaction template* when all constraints in Γ are equalities. We denote these sets by **VarTemp** and **EqTemp**, respectively. For an isolation level $\mathcal{I}$ and a class of transaction templates $\mathcal{C}$, T-ROBUSTNESS($\mathcal{C}$,$\mathcal{I}$) is the problem to decide if a given set of transaction templates $\mathcal{P} \in \mathcal{C}$ is robust against $\mathcal{I}$. When $\mathcal{C}$ is the class of all transaction templates, we simply write T-ROBUSTNESS($\mathcal{I}$).

▶ **Theorem 9** ([17]). T-ROBUSTNESS(***VarTemp***,*RC) is decidable in* PTIME.

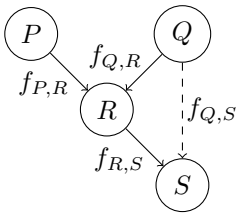


Figure 4 Acyclic schema graph for schema $(\{P, Q, R, S\}, \{f_{P,R}, f_{Q,R}, f_{R,S}, f_{Q,S}\})$. If we remove function name $f_{Q,S}$ (dashed edge), the resulting schema graph is a multi-tree.

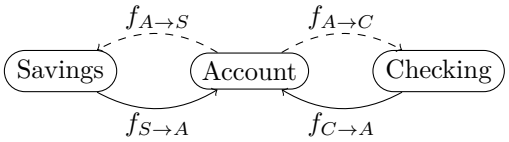


Figure 5 Schema graph for the SmallBank benchmark. The dashed edges correspond to the multi-tree schema graph for the schema restricted to $f_{A\to S}$ and $f_{A\to C}$.

4 Robustness for Templates

We start out with a negative result and show that the robustness problem in its most general form is undecidable (even when disequalities are not allowed). The proof is a reduction from *Post's Correspondence Problem (PCP)* [16] and relies on cyclic dependencies between functional constraints. The proof can be found in the full version of this paper [18] and is quite elaborate but the basic intuition is simple: the counterexample split schedule will build up the two strings that need to be generated by the PCP instance by repeated application of functional constraints.

▶ **Theorem 10.** T-ROBUSTNESS*(**EqTemp**,RC) is undecidable.*

It might be tempting to relate the above result to the undecidability of the implication problem for functional and inclusion dependencies [11]. Functional constraints indeed allow to define inclusion dependencies (as in the SmallBank example) but they always relate complete tuples and are not suited to define functional dependencies. Furthermore, the proof of Theorem 10 makes use of only unary relations, for which the implication problem for functional dependencies and inclusion dependencies is known to be decidable.

To obtain decidable fragments, we introduce restrictions on the structure of functional constraints. The *schema graph* $SG(\mathsf{Rels}, \mathsf{Funcs})$ of a schema $(\mathsf{Rels}, \mathsf{Funcs})$ is a directed multigraph having the relations in Rels as nodes, and in which there are as many edges from a node $R \in \mathsf{Rels}$ to node $S \in \mathsf{Rels}$ as there are functions $f \in \mathsf{Funcs}$ with $dom(f) = R$ and $range(f) = S$. We say that a schema $(\mathsf{Rels}, \mathsf{Funcs})$ is *acyclic* if the multigraph $SG(\mathsf{Rels}, \mathsf{Funcs})$ is acyclic and that it is a *multi-tree* if there is at most one directed path between any two nodes in $SG(\mathsf{Rels}, \mathsf{Funcs})$.

▶ **Example 11.** Consider the schema $(\{P, Q, R, S\}, \{f_{P,R}, f_{Q,R}, f_{R,S}\})$ with $dom(f_{i,j}) = i$ and $range(f_{i,j}) = j$ for each function $f_{i,j}$. The corresponding schema graph with solid lines is given in Figure 4. This schema is a multi-tree, as there is at most one path between any pair of nodes. Notice that the definition of a multi-tree is more general than a forest, as a node can still have multiple parents (e.g., node R in our example). Adding the function name $f_{Q,S}$ with $dom(f_{Q,S}) = Q$ and $range(f_{Q,S}) = S$ results in the schema graph given in Figure 4 that is still acyclic, but no longer a multi-tree as there are now two paths from Q to S. ◻

The schema graph constructed in the proof of Theorem 10 contains several cycles (we refer to [18] for a visualization of the constructed schema graph). We consider in Section 5 robustness for a fragment where a restricted form of cycles in the schema graph is allowed but where additional constraints on the templates are assumed. We consider robustness for acyclic schema graphs in Section 6.

5 Robustness for Templates admitting Multi-Tree Bijectivity

We say that a set of transaction templates $\mathcal{P}$ over a schema $(\mathsf{Rels}, \mathsf{Funcs})$ admits *multi-tree bijectivity* if a disjoint partitioning of Funcs in pairs $(f_1, g_1), (f_2, g_2), \ldots, (f_n, g_n)$ exists such that $dom(f_i) = range(g_i)$ and $dom(g_i) = range(f_i)$ for every pair of function names (f_i, g_i); every schema graph $SG(\mathsf{Rels}, \{h_1, h_2, \ldots, h_n\})$ over the schema restricted to function names $\{h_1, h_2, \ldots, h_n\}$ (with $h_i = f_i$ or $h_i = g_i$) is a multi-tree; and, for every pair of function names (f_i, g_i) and for every pair of variables $\mathtt{X}, \mathtt{Y}$ occurring in a template $\tau_j \in \mathcal{P}$, we have $f_i(\mathtt{X}) = \mathtt{Y} \in \Gamma_j$ iff $g_i(\mathtt{Y}) = \mathtt{X} \in \Gamma_j$. Intuitively, we can think of f_i as a bijective function, with g_i its inverse. We denote the class of all sets of templates admitting multi-tree bijectivity by **MTBTemp**. The SmallBank benchmark discussed in Section 2 is in **MTBTemp**, witnessed by the partitioning $\{(f_{A\to C}, f_{C\to A}), (f_{A\to S}, f_{S\to A})\}$. For example, the schema graph restricted to $f_{A\to C}$ and $f_{A\to S}$ is a tree and therefore also a multi-tree, as illustrated in Figure 5.

The next theorem allows disequalities whereas Theorem 10 does not require them.

▶ **Theorem 12.** T-ROBUSTNESS*(**MTBTemp**,RC) is decidable in* NLOGSPACE.

The approach followed in the proof of Theorem 12 is to repeatedly pick a transaction template while maintaining an overall consistent variable mapping in search for a counter-example multiversion split schedule that by Theorem 7 suffices to show that robustness does not hold. The main challenge is to show that a variable mapping consistent with all functional constraints can be maintained in logarithmic space and that all requirements for a multiversion split schedule can be verified in NLOGSPACE.

Central to our approach is a generalization of conflicting operations. Let $\mathcal{P}$ be a set of transaction templates. For τ_i and τ_j in $\mathcal{P}$, we say that an operation $o_i \in \tau_i$ is *potentially conflicting* with an operation $o_j \in \tau_j$ if o_i and o_j are operations over a variable of the same type, and at least one of the following holds:

- $\text{WriteSet}(o_i) \cap \text{WriteSet}(o_j) \neq \emptyset$ (potentially ww-conflicting);
- $\text{WriteSet}(o_i) \cap \text{ReadSet}(o_j) \neq \emptyset$ (potentially wr-conflicting); or
- $\text{ReadSet}(o_i) \cap \text{WriteSet}(o_j) \neq \emptyset$ (potentially rw-conflicting).

Intuitively, potentially conflicting operations lead to conflicting operations when the variables of these operations are mapped to the same tuple by a variable assignment. In analogy to conflicting quadruples over a set of transactions as in Definition 6, we consider *potentially conflicting quadruples* $(\tau_i, o_i, p_j, \tau_j)$ over $\mathcal{P}$ with $\tau_i, \tau_j \in \mathcal{P}$, and $o_i \in \tau_i$ an operation that is potentially conflicting with an operation $p_j \in \tau_j$. For a sequence of potentially conflicting quadruples $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ over $\mathcal{P}$, we write $\text{Trans}(D)$ to denote the set $\{\tau_1, \ldots, \tau_m\}$ of transaction templates mentioned in D. For ease of exposition, we assume a variable renaming such that any pair of templates in $\text{Trans}(D)$ uses a disjoint set of variables.[5] The sequence D induces a sequence of conflicting quadruples $C = (T_1, b_1, a_2, T_2), \ldots, (T_m, b_m, a_1, T_1)$ by applying a variable assignment μ_i to each τ_i in $\text{Trans}(D)$. We call such a set of variable assignments simply a *variable mapping* for D, denoted $\bar{\mu}$, and write $\bar{\mu}(D) = C$. For a variable $\mathtt{X}$ occurring in a template τ_i, we write $\bar{\mu}(\mathtt{X})$ as a shorthand notation for $\mu_i(\mathtt{X})$, with μ_i the variable assignment over τ_i in $\bar{\mu}$. This is well-defined as all templates in $\text{Trans}(D)$ are variable-disjoint. Furthermore, $\bar{\mu}(var(o_i)) = \bar{\mu}(var(p_j))$ for each potentially

[5] To be formally correct, the latter would require to add every such variable-renamed template to $\mathcal{P}$ creating a larger set $\mathcal{P}'$. This does not influence the complexity of Theorem 12 as $\text{Trans}(D)$ nor $\mathcal{P}'$ are used in the algorithm. Their only purpose is to reason about properties of $\bar{\mu}$.

conflicting quadruple $(\tau_i, o_i, p_j, \tau_j)$ in D as otherwise the induced quadruple (T_i, b_i, a_j, T_j) is not a valid conflicting quadruple in C. We say that a variable mapping $\bar{\mu}$ is admissible for a database $\mathbf{D}$ if every variable assignment μ_i in $\bar{\mu}$ is admissible for $\mathbf{D}$.

A basic insight is that if there is a multiversion split schedule s for some C over a set of transactions $\mathcal{T}$ consistent with $\mathcal{P}$ and a database $\mathbf{D}$, then there is a sequence of potentially conflicting quadruples D such that $\bar{\mu}(D) = C$ for some $\bar{\mu}$. We will verify the existence of such a C, satisfying the properties of Definition 6, by nondeterministically constructing D on-the-fly together with a mapping $\bar{\mu}$. We show in Lemma 14 that when $\mathcal{P} \in \mathbf{MTBTemp}$, $\bar{\mu}$ is a collection of disjoint type mappings (that map variables of the same type to the same tuple) such that variables that are "connected" in D (in a way that we will make precise next) are mapped using the same type mapping. Lemma 15 then shows that already a constant number of those type mappings suffice.

We introduce the necessary notions to capture when two variables are connected in D. We can think of equality constraints $\mathtt{Y} = f(\mathtt{X})$ in a template τ as constraints on the possible variable assignments μ for τ when a database $\mathbf{D}$ is given. Indeed, if we fix $\mu(\mathtt{X})$ to a tuple in $\mathbf{D}$, then $\mu(\mathtt{Y}) = f^{\mathbf{D}}(\mu(\mathtt{X}))$ is immediately implied. These constraints can cause a chain reaction of implications. If for example $\mathtt{Z} = g(\mathtt{Y})$ is a constraint in τ as well, then $\mu(\mathtt{X})$ immediately implies $\mu(\mathtt{Z}) = g^{\mathbf{D}}(f^{\mathbf{D}}(\mu(\mathtt{X})))$. We formalize this notion of implication next. We use sequences of function names $F = f_1 \cdots f_n$, denoting the empty sequence as ε and the concatenation of two sequences F and G by $F \cdot G$. For two variables $\mathtt{X}, \mathtt{Y}$ occurring in a template τ and a (possibly empty) sequence of function names F, we say that $\mathtt{X}$ *implies* $\mathtt{Y}$ *by* F *in* τ, denoted $\mathtt{X} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$, if $\mathtt{X} = \mathtt{Y}$ and $F = \varepsilon$ or if there is a variable $\mathtt{Z}$ such that $\mathtt{Y} = f(\mathtt{Z})$ is a constraint in τ, $\mathtt{X} \overset{F'}{\rightsquigarrow}_\tau \mathtt{Z}$ and $F = F' \cdot f$. We next extend the notions of implication to sequences of potentially conflicting quadruples. Let $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ be a sequence of potentially conflicting quadruples, and let $\mathtt{X}$ and $\mathtt{Y}$ be two variables occurring in templates τ_i and τ_j in $\mathrm{Trans}(D)$, respectively. Then $\mathtt{X}$ *implies* $\mathtt{Y}$ *by a sequence of function names* F *in* D, denoted $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ if

- $i = j$ and $\mathtt{X} \overset{F}{\rightsquigarrow}_{\tau_i} \mathtt{Y}$ (implication within the same template);
- $F = \varepsilon$ and $(\tau_i, o_i, p_j, \tau_j)$ or $(\tau_j, o_j, p_i, \tau_i)$ is a potentially conflicting quadruple in D with o_i (respectively p_i) an operation over $\mathtt{X}$ and p_j (respectively o_j) an operation over $\mathtt{Y}$ (implication between templates, notice that $\mathtt{X} \overset{\varepsilon}{\rightsquigarrow}_D \mathtt{Y}$ iff $\mathtt{Y} \overset{\varepsilon}{\rightsquigarrow}_D \mathtt{X}$); or
- there exists a variable $\mathtt{Z}$ such that $\mathtt{X} \overset{F_1}{\rightsquigarrow}_D \mathtt{Z}$ and $\mathtt{Z} \overset{F_2}{\rightsquigarrow}_D \mathtt{Y}$ with $F = F_1 \cdot F_2$.

Two variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in $\mathrm{Trans}(D)$ are *connected in* D, denoted $\mathtt{X} \approx_D \mathtt{Y}$, if $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ or $\mathtt{Y} \overset{F}{\rightsquigarrow}_D \mathtt{X}$, or if there is a variable $\mathtt{Z}$ with $\mathtt{X} \approx_D \mathtt{Z}$ and either $\mathtt{Z} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ or $\mathtt{Y} \overset{F}{\rightsquigarrow}_D \mathtt{Z}$ for some sequence F. Furthermore, two variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in a template τ are *connected in* τ, denoted $\mathtt{X} \approx_\tau \mathtt{Y}$, if $\mathtt{X} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$ or $\mathtt{Y} \overset{F}{\rightsquigarrow}_\tau \mathtt{X}$, or if there is a variable $\mathtt{Z}$ with $\mathtt{X} \approx_\tau \mathtt{Z}$ and either $\mathtt{Z} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$ or $\mathtt{Y} \overset{F}{\rightsquigarrow}_\tau \mathtt{Z}$ for some sequence F. These definitions of connectedness can be trivially extended to operations over variables: two operations in D (respectively τ) are connected in D (respectively τ) if they are over variables that are connected in D (respectively τ). When F is not important we drop it from the notation. For instance, we denote by $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ that there is an F with $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$.

▶ **Lemma 13.** *Let D be a sequence of potentially conflicting quadruples over $\mathcal{P} \in \mathbf{MTBTemp}$. Then $\mathtt{X} \approx_D \mathtt{Y}$ implies $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ and $\mathtt{Y} \rightsquigarrow_D \mathtt{X}$. Furthermore, if type$(\mathtt{X})$ = type$(\mathtt{Y})$ then $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$ for every variable mapping $\bar{\mu}$ for D that is admissible for some database $\mathbf{D}$.*

It follows from Lemma 13 that, if we group connected variables, then the same tuple is assigned to all variables of the same type in this group. We encode this choice of tuples for variables through (total) functions $c : \mathsf{Rels} \to \mathbf{Tuples}$ that we call *type mappings* and which

map a relation onto a particular tuple of that relation's type. For instance, in SmallBank, a type mapping c is determined by an Account tuple $\mathtt{a}$, a Savings tuple $\mathtt{s}$, and a Checking tuple $\mathtt{c}$. The following lemma makes explicit how $\bar{\mu}$ can be decomposed into type mappings such that connected variables use the same type mapping and disequalities enforce the use of different type mappings.

▶ **Lemma 14.** *For a multiversion split schedule s based on a sequence of conflicting quadruples C over a set of transactions $\mathcal{T}$ consistent with a $\mathcal{P} \in \boldsymbol{MTBTemp}$ and a database $\boldsymbol{D}$, let $\bar{\mu}$ be the variable mapping for a sequence of potentially conflicting quadruples D over $\mathcal{P}$ with $\bar{\mu}(D) = C$. Then, a set $\mathcal{S}$ of type mappings over disjoint ranges and a function $\varphi_{\mathcal{S}} : \mathbf{Var} \to \mathcal{S}$ exist with:*

- $\bar{\mu}(\mathtt{X}) = c(type(\mathtt{X}))$ *for every variable* $\mathtt{X}$*, with* $c = \varphi_{\mathcal{S}}(\mathtt{X})$*;*
- $\varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Y})$ *whenever* $\mathtt{X} \approx\!\!\approx_D \mathtt{Y}$*; and,*
- $\varphi_{\mathcal{S}}(\mathtt{X}) \neq \varphi_{\mathcal{S}}(\mathtt{Y})$ *for every constraint* $\mathtt{X} \neq \mathtt{Y}$ *occurring in a template* $\tau \in Trans(D)$*.*

From $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ and $\varphi_{\mathcal{S}}$ as in Lemma 14 we can derive a sequence of quintuples $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1}), \ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ such that $c_{o_i} = \varphi_{\mathcal{S}}(var(o_i))$ and $c_{p_i} = \varphi_{\mathcal{S}}(var(p_i))$ for $i \in [1, m]$. Intuitively, this sequence of quintuples can be used to reconstruct the original multiversion split schedule s. The next lemma shows that we can decide robustness against RC over a set of transaction templates admitting multi-tree bijectivity by searching for a specific sequence of quintuples over at most four type mappings.

▶ **Lemma 15.** *Let $\mathcal{P} \in \boldsymbol{MTBTemp}$ and let $\mathcal{S} = \{c_1, c_2, c_3, c_4\}$ be a set consisting of four type mappings with disjoint ranges. Then, $\mathcal{P}$ is not robust against RC iff there is a sequence of quintuples $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1}), \ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ with $m \geq 2$ such that for each quintuple $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ in E:*

1. *o_i and p_i are operations in τ_i, and $c_{o_i}, c_{p_i} \in \mathcal{S}$;*
2. *$\mathtt{X}_i \not\approx\!\!\approx_{\tau_i} \mathtt{Y}_i$ for each constraint $\mathtt{X}_i \neq \mathtt{Y}_i$ in τ_i;*
3. *$c_{o_i} = c_{p_i}$ if $o_i \approx\!\!\approx_{\tau_i} p_i$;*
4. *$c_{o_i} \neq c_{p_i}$ if there is a constraint $\mathtt{X}_i \neq \mathtt{Y}_i$ in τ_i with $\mathtt{X}_i \approx\!\!\approx_{\tau_i} var(o_i)$ and $\mathtt{Y}_i \approx\!\!\approx_{\tau_i} var(p_i)$;*
5. *if $i \neq 1$ and $c_{q_i} = c_{q_1}$ for some $q_i \in \{o_i, p_i\}$ and $q_1 \in \{o_1, p_1\}$, then there is no operation o_i' in τ_i potentially ww-conflicting with an operation o_1' in $\mathsf{prefix}_{o_1}(\tau_1)$ with $var(o_i') \approx\!\!\approx_{\tau_i} var(q_i)$ and $var(o_1') \approx\!\!\approx_{\tau_1} var(q_1)$.*

Furthermore, for each pair of adjacent quintuples $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ and $(\tau_j, o_j, c_{o_j}, p_j, c_{p_j})$ in E with $j = i + 1$, or $i = m$ and $j = 1$:

6. *o_i is potentially conflicting with p_j and $c_{o_i} = c_{p_j}$;*
7. *if $i = 1$ and $j = 2$, then o_1 is potentially rw-conflicting with p_2; and*
8. *if $i = m$ and $j = 1$, then $o_1 <_{\tau_1} p_1$ or o_m is potentially rw-conflicting with p_1.*

The items have the following meaning: (2) τ_i is satisfiable; (3) connected operations are assigned the same type mapping; (4) variables connected through an inequality are assigned a different type mapping; (5) $\varphi_{\mathcal{S}}$ only assigns the same type mapping to o_1 or p_1 in τ_1 and o_i or p_i in τ_i if it does not introduce a dirty write in the resulting multiversion split schedule (cf. Condition (1) in Definition 6); (6) each pair of variables in operations used for conflicts are assigned the same type mapping; (7, 8) the operations used for conflicts between τ_1, τ_2 and τ_m are restricted to satisfy respectively Condition (3) and (2) in Definition 6 in the resulting multiversion split schedule.

The characterization for T-ROBUSTNESS(**MTBTemp**,RC) in Lemma 15 implies an NLOGSPACE algorithm guessing the counterexample sequence E, thereby proving Theorem 12. Indeed, the algorithm guesses the sequence of quintuples E, verifying all conditions for each newly guessed quintuple while only requiring logarithmic space. Notice in particular that

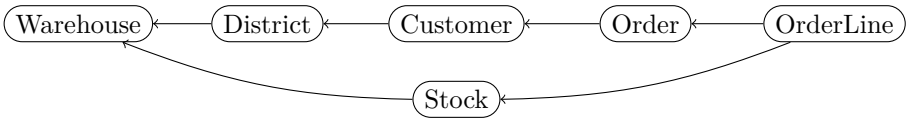


Figure 6 Acyclic schema graph for the TPC-C benchmark.

we only need to keep track of two other quintuples when verifying all conditions for the newly guessed quintuple, namely the first quintuple over τ_1 and the quintuple immediately preceding the newly guessed one. As usual, we can think of the encoding of templates and operations mentioned in each quintuple as pointers referring to the corresponding templates and operations on the input tape. Furthermore, we do not encode the four type mappings explicitly as such a representation of a mapping might require polynomial space. Since we are only interested in (dis)equality between type mappings, an encoding where these four type mappings are represented by four arbitrary strings of constant size suffices. More details can be found in [18].

6 Robustness for Templates over Acyclic Schemas

We denote by **AcycTemp** the class of all sets of transaction templates over acyclic schemas. As a concrete example, the schema graph for the TPC-C benchmark is given in Figure 6. Since this schema graph does not contain any cycles, the TPC-C benchmark is situated within **AcycTemp**. Notice in particular how this acyclic schema graph corresponds to the hierarchical structure of many-to-one relationships inherent to the schema for this benchmark. For example, every orderline belongs to exactly one order, and every order is related to exactly one customer, but the opposite is never true (i.e., a customer can be related to multiple orders, each of which can be related to multiple orderlines). In general, the results presented in this section can be applied to all workloads over schemas with such a hierarchical structure.

▶ **Theorem 16.** T-ROBUSTNESS*(**AcycTemp**,RC) is decidable in* EXPSPACE.

We provide some intuition for the proof. For a given acyclic schema graph SG, $R \overset{F}{\rightsquigarrow}_{SG} S$ denotes the directed path from node R to node S in SG with F the sequence of edge labels on the path. The next lemma relates implication between variables to paths in SG.

▶ **Lemma 17.** *Let D be a sequence of potentially conflicting quadruples over a set of transaction templates $\mathcal{P} \in$ **AcycTemp**. For every pair of variables $\mathtt{X}, \mathtt{Y}$ occurring in $Trans(D)$, if $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$, then $type(\mathtt{X}) \overset{F}{\rightsquigarrow}_{SG} type(\mathtt{Y})$, with SG the corresponding schema graph.*

Notice that an assignment of a tuple to a variable $\mathtt{X}$ determines the tuples assigned to all variables $\mathtt{Y}$ with $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ for some sequence of function names F. From Lemma 17 it follows that each such implied tuple is witnessed by a path in the corresponding schema graph SG. Therefore, the maximal number of different tuples implied by $\mathtt{X}$ corresponds to the number of paths in SG starting in type($\mathtt{X}$), which is finite when SG is acyclic. Because there can be multiple paths between nodes in the schema graph, it is no longer the case as in the previous section that variables of the same type connected in D must be assigned the same value. So, instead of using type mappings, we introduce *tuple-contexts* to represent the sets of all tuples implied by the assignment of a given variable. Formally, a *tuple-context for a type* $R \in \mathsf{Rels}$ is a function from paths with source R in $SG(\mathsf{Rels}, \mathsf{Funcs})$ to tuples in **Tuples** of the appropriate type. That is, for each tuple-context c for type R and for each path $R \overset{F}{\rightsquigarrow}_{SG} S$ in SG, $\text{type}(c(R \overset{F}{\rightsquigarrow}_{SG} S)) = S$.

Similar to Lemma 14, we show that we can represent a counterexample schedule based on D by assigning a tuple-context to each variable in Trans(D), taking special care when assigning contexts to variables connected in D to make sure that they are properly related to each other. For this, we introduce a (partial) function $\varphi_{\mathcal{A}} : \mathbf{Var} \to \mathcal{A}$ mapping (a subset of) variables in Trans(D) to tuple-contexts in $\mathcal{A}$ (for $\mathcal{A}$ a set of tuple-contexts)and refer to it as a *(partial) context assignment for D over $\mathcal{A}$*. In a sequence of lemma's, we show that $\varphi_{\mathcal{A}}$ can always be expanded into a total function and an approach based on enumeration of quintuples analogous to Lemma 15 suffices to decide robustness. A major difference with the previous section is that there is no longer a constant bound on the number of tuple-contexts that are needed and consistency between tuple-contexts in connected variables needs to be maintained. A full proof can be found in [18].

Next, we consider restrictions that lower the complexity. To this end, we say that two variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in a transaction template τ are *equivalent in* τ, denoted $\mathtt{X} \equiv_\tau \mathtt{Y}$ if

- $\mathtt{X} = \mathtt{Y}$;
- there exists a pair of variables $\mathtt{Z}$ and $\mathtt{W}$ in τ and a sequence of function names F with $\mathtt{Z} \equiv_\tau \mathtt{W}$, $\mathtt{Z} \overset{F}{\rightsquigarrow}_\tau \mathtt{X}$ and $\mathtt{W} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$; or
- there exists a variable $\mathtt{Z}$ with $\mathtt{X} \equiv_\tau \mathtt{Z}$ and $\mathtt{Y} \equiv_\tau \mathtt{Z}$.

Then, a transaction template τ is *restricted* if for every combination of variables $\mathtt{X}, \mathtt{Y}, \mathtt{W}, \mathtt{Z}$ in τ with $\mathtt{X} \rightsquigarrow_\tau \mathtt{W}$ and $\mathtt{Y} \rightsquigarrow_\tau \mathtt{Z}$, either $\mathtt{W} \equiv_\tau \mathtt{Z}$, $\mathtt{W} \rightsquigarrow_\tau \mathtt{Z}$ or $\mathtt{Z} \rightsquigarrow_\tau \mathtt{W}$. We denote by **AcycResTemp** the class of all sets of restricted transaction templates over acyclic schemas.

▶ **Theorem 18.** 1. T-ROBUSTNESS*(**AcycResTemp**,RC) is decidable in* EXPTIME.

2. T-ROBUSTNESS*(**AcycTemp**,RC) is decidable in* PSPACE *when the number of paths between any two nodes in the schema graph is bounded by a constant k.*

Regarding (1), all templates in TPC-C with the exception of NewOrder are restricted. Regarding (2), when the schema graph is a multi-tree then $k = 1$ and for TPC-C $k = 2$ (recall that in general there can be an exponential number of paths), leading to a more practical algorithm for robustness in those cases.

7 Related Work

Transaction Programs

Previous work on static robustness testing [13, 3] for transaction programs is based on the following key insight: when a *schedule* is not serializable, then the dependency graph constructed from that schedule contains a cycle satisfying a condition specific to the isolation level at hand (*dangerous structure* for SNAPSHOT ISOLATION and the presence of a *counterflow edge* for RC). That insight is extended to a workload of *transaction programs* through the construction of a so-called static dependency graph where each program is represented by a node, and there is a conflict edge from one program to another if there can be a schedule that gives rise to that conflict. The absence of a cycle satisfying the condition specific to that isolation level then guarantees robustness while the presence of a cycle does not necessarily imply non-robustness.

Other work studies robustness within a framework for uniformly specifying different isolation levels in a declarative way [8, 7, 9]. A key assumption here is *atomic visibility* requiring that either all or none of the updates of each transaction are visible to other transactions. These approaches aim at higher isolation levels and cannot be used for RC, as RC does not admit *atomic visibility*.

Transaction Templates

The static robustness approach based on transaction templates [17] differs in two ways. First, it makes more underlying assumptions explicit within the formalism of transaction templates (whereas previous work departs from the static dependency graph that should be constructed in some way by the dba). Second, it allows for a decision procedure that is sound and complete for robustness testing against RC, allowing to detect larger subsets of transactions to be robust [17].

The formalisation of transactions and conflict serializability in [17] and this paper is based on [12], generalized to operations over attributes of tuples and extended with `U`-operations that combine `R`- and `W`-operations into one atomic operation. These definitions are closely related to the formalization presented by Adya et al. [1], but we assume a total rather than a partial order over the operations in a schedule. There are also a few restrictions to the model: there needs to be a fixed set of read-only attributes that cannot be updated and which are used to select tuples for update. The most typical example of this are primary key values passed to transaction templates as parameters. The inability to update primary keys is not an important restriction in many workloads, where keys, once assigned, never get changed, for regulatory or data integrity reasons.

In [17], a PTIME decision procedure is obtained for robustness against RC for templates without functional constraints and the present paper improves that result to NLOGSPACE. In addition, an experimental study was performed showing how an approach based on robustness and making transactions robust through promotion can improve transaction throughput.

Transactions

Fekete [12] is the first work that provides a necessary and sufficient condition for deciding robustness against SNAPSHOT ISOLATION for a workload of concrete transactions (not transaction programs). That work provides a characterization for acceptable allocations when every transaction runs under either SNAPSHOT ISOLATION or strict two-phase locking (S2PL). The allocation then is acceptable when every possible execution respecting the allocated isolation levels is serializable. As a side result, this work indirectly provides a necessary and sufficient condition for robustness against SNAPSHOT ISOLATION, since robustness against SNAPSHOT ISOLATION holds iff the allocation where each transaction is allocated to SNAPSHOT ISOLATION is acceptable. Ketsman et al. [14] provide full characterisations for robustness against READ COMMITTED and READ UNCOMMITTED under lock-based semantics. In addition, it is shown that the corresponding decision problems are complete for coNP and LOGSPACE, respectively, which should be contrasted with the polynomial time characterization obtained in [17] for robustness against *multiversion* read committed.

8 Conclusion

This paper falls within a more general research line investigating how transaction throughput can be improved through an approach based on robustness testing that can be readily applied without making any changes to the underlying database system. As argued in Section 2, incorporating functional constraints can detect larger sets of templates to be robust and requires less `R`-operations to be promoted to `U`-operations. In future work, we plan to look at lower bounds, restrictions that lower complexity, and consider other referential integrity constraints to further enlarge the modelling power of transaction templates.

References

1 Atul Adya, Barbara Liskov, and Patrick E. O'Neil. Generalized isolation level definitions. In *ICDE*, pages 67–78, 2000.

2 Mohammad Alomari, Michael Cahill, Alan Fekete, and Uwe Rohm. The cost of serializability on platforms that use snapshot isolation. In *ICDE*, pages 576–585, 2008.

3 Mohammad Alomari and Alan Fekete. Serializable use of read committed isolation level. In *AICCSA*, pages 1–8, 2015.

4 Sidi Mohamed Beillahi, Ahmed Bouajjani, and Constantin Enea. Checking robustness against snapshot isolation. In *CAV*, pages 286–304, 2019.

5 Sidi Mohamed Beillahi, Ahmed Bouajjani, and Constantin Enea. Robustness against transactional causal consistency. In *CONCUR*, pages 1–18, 2019.

6 Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, pages 1–10, 1995.

7 Giovanni Bernardi and Alexey Gotsman. Robustness against consistency models with atomic visibility. In *CONCUR*, pages 7:1–7:15, 2016.

8 Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. A framework for transactional consistency models with atomic visibility. In *CONCUR*, pages 58–71, 2015.

9 Andrea Cerone and Alexey Gotsman. Analysing snapshot isolation. *J.ACM*, 65(2):1–41, 2018.

10 Andrea Cerone, Alexey Gotsman, and Hongseok Yang. Algebraic Laws for Weak Consistency. In *CONCUR*, pages 26:1–26:18, 2017.

11 Ashok K. Chandra and Moshe Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, 14(3):671–677, 1985.

12 Alan Fekete. Allocating isolation levels to transactions. In *PODS*, pages 206–215, 2005.

13 Alan Fekete, Dimitrios Liarokapis, Elizabeth J. O'Neil, Patrick E. O'Neil, and Dennis E. Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.

14 Bas Ketsman, Christoph Koch, Frank Neven, and Brecht Vandevoort. Deciding robustness for lower SQL isolation levels. In *PODS*, pages 315–330, 2020.

15 Christos H. Papadimitriou. *The Theory of Database Concurrency Control*. Computer Science Press, 1986.

16 Emil L. Post. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.*, pages 264–268, 1946.

17 Brecht Vandevoort, Bas Ketsman, Christoph Koch, and Frank Neven. Robustness against read committed for transaction templates. *PVLDB*, 14(11):2141–2153, 2021.

18 Brecht Vandevoort, Bas Ketsman, Christoph Koch, and Frank Neven. Robustness against read committed for transaction templates with functional constraints (full version), 2022. URL: `https://arxiv.org/abs/2201.05021`.

A Dyadic Simulation Approach to Efficient Range-Summability

Jingfan Meng
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Huayi Wang
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Jun Xu
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Mitsunori Ogihara
Department of Computer Science, University of Miami, Coral Gables, FL, USA

Abstract

Efficient range-summability (ERS) of a long list of random variables is a fundamental algorithmic problem that has applications to three important database applications, namely, data stream processing, space-efficient histogram maintenance (SEHM), and approximate nearest neighbor searches (ANNS). In this work, we propose a novel dyadic simulation framework and develop three novel ERS solutions, namely Gaussian-dyadic simulation tree (DST), Cauchy-DST and Random Walk-DST, using it. We also propose novel rejection sampling techniques to make these solutions computationally efficient. Furthermore, we develop a novel k-wise independence theory that allows our ERS solutions to have both high computational efficiencies and strong provable independence guarantees.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Random number generation

Keywords and phrases fast range-summation, locality-sensitive hashing, rejection sampling

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.17

Related Version *Previous Version*: https://arxiv.org/abs/2109.06366

Funding This material is based upon work supported by the National Science Foundation under Grant No. CNS-1909048, CNS-2007006, CNS-2051800, and by Keysight Technologies under Grant No. BG005054.

1 Introduction

In this work, we propose *dyadic simulation*, a novel solution framework to a fundamental algorithmic problem that has applications to three important database applications: data stream processing [8], space-efficient histogram maintenance (SEHM) [7, 11] and approximate nearest neighbor searches (ANNS) [24]. This algorithmic problem, called *efficient range-summability* (ERS) of random variables (RVs) [5, 17], can be stated as follows. Let $X_0, X_1, \cdots, X_{U-1}$ be a list of i.i.d. RVs, where the (index) universe size U is typically a large number (say $U = 2^{64}$). Given a range $[a, b) \triangleq \{a, a+1, \cdots, b-1\}$ that lies in $[0, U)$, we need to compute $S[a, b) \triangleq \sum_{i=a}^{b-1} X_i$, the sum of the RVs $X_a, X_{a+1}, \cdots, X_{b-1}$ in the range. A straightforward but naive solution to this problem, which follows an intuitive "bottom-up" approach, is to generate RVs $X_a, X_{a+1}, \cdots, X_{b-1}$ individually and then add them up. This solution, however, has a time complexity of $O(b-a)$, which is inefficient computationally when the range length $b-a$ is large. In contrast, an efficient solution should be able to do so with only $O(\text{polylog}(b-a))$ time complexity. Indeed, all existing ERS solutions [2, 5, 17, 7] have $O(\log(b-a))$ time complexity.

Each such ERS solution generates a range-sum $S[a,b)$ or an underlying RV X_i in a very different way than the naive solution. This difference however does not matter since an ERS solution is considered correct as long as it satisfies two requirements: consistency and correct distribution. The consistency requirement is that, given any *outcome* ω in the *sample space* Ω (in probability theory terms), the realization of $S[a,b)$ associated with the outcome ω must be equal to $\sum_{i=a}^{b-1} X_i(\omega)$, where $X_i(\omega)$ is the realization of X_i associated with the same outcome ω. The correct distribution requirement is that the underlying RVs thus generated are ideally i.i.d. with distribution X.

1.1 Our Dyadic Simulation Approach

For ease of presentation, we make two harmless simplifying assumptions. The first assumption is a typical "computer science" one: The universe size U is a power of 2. This assumption can always be fulfilled by increasing U to at most $2U$. The second assumption is that $[a,b)$ is a dyadic range in the sense there exist integers $j \geq 0$ and $i \geq 0$ such that $a = j \cdot 2^i$ and $b = (j+1) \cdot 2^i$. It suffices for our solution to work for any dyadic range since any non-dyadic range can be split into at most $2\log_2 U$ dyadic ranges, as we will elaborate in Subsection 2.1.

Unlike the naive solution, our dyadic simulation approach computes $S[a,b)$ in a counter-intuitive "top-down" manner as follows. Its first step is to generate the RV $S[0,U)$, the range-sum of the entire universe. If we denote the distribution of each underlying RV X_i as X, then $S[0,U)$ has distribution X^{*U}, where, for any $n > 1$, X^{*n} denotes the n^{th} convolution power of X. When X is one of a few nice distributions, the distribution X^{*U} can be analytically derived and also takes a nice form; in this case, it is straightforward to generate $S[0,U)$. For example, when X is standard Gaussian distribution $\mathcal{N}(0,1)$, then X^{*U} is $\mathcal{N}(0,U)$.

The rest of dyadic simulation proceeds as follows. If $[a,b)$ is the same as $[0,U)$, then the ERS problem is solved. Otherwise, we split $S[0,U)$ into two half-range-sums $S[0,U/2) + S[U/2,U)$, such that RVs $S[0,U/2)$ and $S[U/2,U)$ are (mutually) independent and each has distribution $X^{*(U/2)}$. While this may sound wishful thinking, we will show in Section 2 that it is always mathematically possible and can be done in a computationally efficient manner in some cases.

After the split, we have either $[a,b) \subseteq [0,U/2)$ or $[a,b) \subseteq [U/2,U)$ by Proposition 2.4 in [17], since $[a,b)$, $[0,U/2)$, and $[U/2,U)$ are all dyadic intervals. We then recursively "binary-search" for $[a,b)$ either in the left-half $[0,U/2)$ if $[a,b) \subseteq [0,U/2)$ or in the right-half $[U/2,U)$ if $[a,b) \subseteq [U/2,U)$. It is not hard to verify that after at most $\log_2 U$ such splits we can "find" $[a,b)$ and as a result compute $S[a,b)$. Hence, the time complexity of a dyadic simulation algorithm is $O(\log U)$ splits for generating any dyadic range-sum. Perhaps surprisingly, even for generating any range-sum that is not necessarily dyadic, the time complexity remains $O(\log U)$ splits instead of becoming $O(\log^2 U)$, as we will show in Subsection 2.1.

We can generate any underlying RV X_i via $\log_2 U$ such binary splits, because $X_i \equiv S[i,i+1)$, and $[i,i+1)$ is a dyadic range. We say dyadic simulation takes a "top-down" approach because when all the underlying RVs $X_0, X_1, \cdots, X_{U-1}$ are generated this way, they become the "leaves" (at the "bottom") of the full binary tree of the binary splits involved in generating them. This tree, called *dyadic simulation tree* (DST), will be officially introduced in Subsection 2.1. In this work, we propose novel DST-based solutions to three ERS problems whose underlying RVs have Gaussian, Cauchy, and single-step random walk (RW) (*aka.* Rademacher) distributions, respectively. We also propose novel rejection sampling techniques that make these three solutions, called Gaussian-DST, Cauchy-DST, and RW-DST respectively, computationally efficient. Each binary split operation takes only nanoseconds for Gaussian and 20+ nanoseconds for Cauchy and random walk, as we will show in Subsection 2.6.

All existing ERS solutions were proposed for the single-step random walk distribution $\Pr[X = 1] = \Pr[X = -1] = 0.5$. Here we highlight a key difference between our dyadic simulation approach and these ERS solutions. This difference is a major contribution of this work. The underlying RVs X_0, X_1, $\cdots$, X_{U-1} generated by our dyadic simulation approach are at least empirically independent for all practical purposes. In contrast, those generated by all existing ERS solutions are strongly correlated. For example, in the EH3 scheme proposed in [5, 17], the underlying RVs are approximately 4-wise independent, but all independence beyond 4-wise is completely destroyed. However, in nearly all applications of dyadic simulation that we will describe next, we need these RVs to be at least empirically independent.

A very sketchy idea of dyadic simulation was proposed, in a few sentences, in a theory paper [7] that mainly focused on the aforementioned SEHM problem. Although it was stated in [7] that dyadic simulation can possibly be used for the ERS of Gaussian and Cauchy RVs, no computationally efficient technique was specified in it for binary-splitting a Gaussian or Cauchy RV. We will elaborate on such techniques in Subsections 2.3 and 2.4.

1.2 Independence Guarantees

As we have just explained, each non-leaf node in a DST corresponds to a dyadic range $[a, b)$, whose two children correspond to the two dyadic half ranges $[a, (a+b)/2)$ and $[(a+b)/2, b)$. We will show in Subsection 2.2 that each such non-leaf node, now identified by its corresponding dyadic range say $[a, b)$, is associated with a uniformly random binary string $C_{[a,b)}$ that determines the values of half-range-sums $S[a, (a + b)/2)$ and $S[(a + b)/2, b)$ that the range-sum $S[a, b)$ is split into. Depending on how each $C_{[a,b)}$ is generated, we can obtain various theoretical guarantees concerning how independent the underlying RVs $X_0, X_1, \cdots, X_{U-1}$ are.

Ideally, each such $C_{[a,b)}$ should be a freshly generated RV in the sense that it is independent of all other RVs. If this is the case, then we can prove that, starting with $S[0, U)$ that is distributed as X^{*U}, the U underlying RVs generated through these binary splits are i.i.d. with distribution X. However, this idealized case is impractical when the universe size U is massive, since the value of each freshly generated RV would all have to be remembered (stored in memory) and there can be a massive number of them. In practice, we typically generate each such $C_{[a,b)}$ value (on demand) by applying a hash function $h(\cdot)$ to the dyadic range $[a, b)$. There are two standard choices of such hash functions in the literature. The practical type is "off-the-shelf" random hash functions that can produce a hash value in nanoseconds, such as `wyhash` [25]. Although they provide no theoretical guarantees, they are demonstrated to ensure a level of empirical independence that is good enough for all practical applications [23]. The theoretical type, called k-wise independent hash functions [3, 21, 15], generates $(C_{[a,b)})$'s that are k-wise independent. In this work, we establish a novel k-wise independence theory for DST which shows, among other things, that k-wise independence among $C_{[a,b)}$ values implies k-wise independence among the underlying RVs. Although the latter theoretical guarantee is weaker than the ideal all-wise mutual independence, it leads to rigorous theoretical guarantees that are strong enough for most ERS applications, as we will show in Subsection 1.3.

We note all our DST solutions can use Nisan's pseudorandom generator (PRG) [12], which delivers strong independence guarantees for memory- (state-space-) constrained algorithms. However, Nisan's PRG is quite computationally intensive, and hence has never been implemented and used in practice. Indeed, a key contribution of our k-wise independence theory lies in its ability to satisfy the "theoretical needs" of most ERS applications using k-wise independent hash functions that are much less computationally intensive.

1.3 Applications

In this section, we describe the three aforementioned applications that motivate our DST-based ERS solutions. Since we claim none of them as a contribution of this work, each description here is only detailed enough to explain how an ERS problem arises in it. Furthermore, we will not elaborate on any application in the rest of this paper.

The *first* application is data stream processing, where two of our ERS solutions extend an existing data streaming algorithm suite for efficiently handling *range updates*. We start our introduction with an oversimplified characterization of the data stream model. In this model, the precise system state is comprised of a large number (say U) of counters σ_0, σ_1, $\cdots$, σ_{U-1} whose values are initialized to 0. A data stream is comprised of a large number of data items that can take one of the following two forms: *standard (point-update)* and *range-update*. In a standard data stream, each item, say the t^{th}, in the data stream is in the form (i_t, δ_t). This data item should cause the following update to the precise system state: Counter σ_{i_t} is to be incremented by δ_t, which we call a *point update*. In a range-update data stream, which is more general (than standard data streams), each data item is in the form $([a_t, b_t), \delta_t)$. In this case, for each index i in the range $[a_t, b_t)$, the corresponding counter σ_i needs to be incremented by δ_t, which we call a *range update*. A typical data streaming query is to estimate a certain function of the counter values after the updates caused by all the data items in the data stream are committed to the system state. For example, the L_2-norm and the L_1-norm estimation problems are to estimate the values of $d_2 \triangleq (\sum_{i=0}^{U-1} |\sigma_i|^2)^{1/2}$ (the L_2-norm of the system state) and $d_1 \triangleq \sum_{i=0}^{U-1} |\sigma_i|$ (the L_1-norm), respectively. Since U is usually too huge for the precise system state to fit in fast memory, a data streaming algorithm has to summarize it into a synopsis data structure called a *sketch*, whose size is much smaller than $O(U)$.

A data streaming algorithm suite, proposed in [8], solves the L_2- and the L_1-norm estimation problems for standard data streams. It employs a Gaussian-sum or Cauchy-sum sketch comprised of $r > 0$ i.i.d. accumulators (viewed as RVs) $A_1, A_2, \cdots, A_r$. Since these accumulators are independent and functionally equivalent, it suffices to describe the point-update procedure for one such accumulator, which we denote as A. A is initialized to 0 at the beginning. Given a point update (i_t, δ_t), A is incremented by $\delta_t X_{i_t}$, where X_{i_t} is a standard Gaussian (for L_2-norm) or Cauchy (for L_1-norm) RV associated with the counter σ_{i_t}. The value of X_{i_t} is fixed after it is generated on-demand for the first time. After the entire data stream has passed, it was shown in [8] that $A = \sum_{i=0}^{U-1} \sigma_i X_i$ has distribution $\mathcal{N}(0, d_2^2)$ or Cauchy$(0, d_1)$ respectively, wherein the parameters d_2^2 and d_1 can be estimated using standard estimators.

This algorithm can handle a range update $([a_t, b_t), \delta_t)$ as follows:

For $i = a_t$ to $b_t - 1$, do $A \leftarrow A + \delta_t X_i$.

However, the time complexity of this update procedure is $O(b_t - a_t)$, which is very high when $b_t - a_t$ is gigantic. In comparison, our Gaussian-DST and Cauchy-DST solutions can process this range update in $O(\log(b_t - a_t))$ time, since the net effect of this range update is to increment A by $\delta_t \cdot (\sum_{i=a_t}^{b_t-1} X_i)$, which is precisely δ_t times the (Gaussian or Cauchy) range-sum $S[a_t, b_t)$.

In [8], the median estimator is used in L_1-norm estimation, in which case all-wise independence of the underlying RVs are needed for a theoretical guarantee. However, for L_2-norm estimation, a mean-estimator is used [8], which is a standard quadratic polynomial of the accumulators $\hat{d}_2^2 = (A_1^2 + A_2^2 + \cdots + A_r^2)/r$. In this case, it can be shown (e.g., using arguments similar to those in Theorem 2.2 in [1]) that the L_2-norm estimator achieves the

same statistical efficiency whether the underlying RVs are 4-wise independent or all-wise independent. According to Theorem 9 (in Section 3), our Gaussian-DST solution guarantees that the underlying RVs are 4-wise independent when it is implemented using $\log_2 U$ 4-wise independent hash functions.

The *second* application is the space-efficient histogram maintenance (SEHM) problem in the data streaming setting, which as mentioned earlier was the focus of [7]. The precise system state to be approximately maintained by a proposed SEHM solution is a scaled probability mass function (pmf) $f(\cdot)$ whose domain is the set of integers $\{0, 1, 2, \cdots, U-1\}$, where the universe U is typically a large (positive) integer; we denote this domain simply as $[0, U)$. This $f(\cdot)$ starts as a zero function, and at any moment τ, $f(\cdot)$ is defined by a stream of point updates before or at τ in the sense each point update (i_τ, δ_τ) causes the value of $f(i_\tau)$ to be incremented by δ_τ. Hence $f(\cdot)$ is a "pmf in motion".

A part of the SEHM problem is to answer the following query. At any given moment τ, the proposed SEHM solution needs to approximately represent the snapshot of $f(\cdot)$ at τ using a good and simple *histogram* function whose domain is also $[0, U)$. Here, a histogram $\mathbf{H}$ is a piecewise-constant function defined by B non-overlapping intervals (buckets) $I_1, I_2, \cdots, I_B$ that comprise $[0, U)$ and B *spline parameters* $\chi_1, \chi_2, \cdots, \chi_B$ that define the height of each bucket, as follows: $\mathbf{H}(i) = \chi_j$ when $i \in I_j$, for $i = 0, 1, \cdots, U-1$. The approximation error of $\mathbf{H}$ (relative to $f(\cdot)$) is defined as the L_2-error $(\sum_{i=0}^{U-1} |\mathbf{H}(i) - f(i)|^2)^{1/2}$ or the L_1-error $\sum_{i=0}^{U-1} |\mathbf{H}(i) - f(i)|$. A histogram $\mathbf{H}$ is called simple when B is small and called good when its approximate error is small.

A subproblem of this query problem is, given a (simple) candidate histogram $\mathbf{H}$, to determine whether it is good in terms of L_2- or L_1-error. It was shown in [7] that the SEHM problem can be solved by maintaining a Gaussian-sum (for the L_2 case) or a Cauchy-sum (for the L_1 case) sketch of $f(\cdot)$. In addition, for solving this subproblem given a candidate histogram $\mathbf{H}$, a Gaussian-sum or Cauchy-sum sketch of $\mathbf{H}$ needs to be computed. Suppose $I_j = [a_j, a_{j+1})$ for $j = 1, 2, \cdots, B$. Then the value of an accumulator A in the sketch of $\mathbf{H}$ takes value $A = \sum_{j=1}^{B} \chi_j S[a_j, a_{j+1})$ (as explained above), where each $S[a_j, a_{j+1})$ is a Gaussian or Cauchy range-sum that needs to be *efficiently computed*. It was shown in [7] that the L_2- or L_1-error of approximating $f(\cdot)$ by $\mathbf{H}$ can be estimated from the difference between the sketches of $f(\cdot)$ and $\mathbf{H}$.

We now shift our attention to the *third* application of ERS: Locality-Sensitive Hashing (LSH) schemes for approximate nearest neighbors searches (ANNS). An ERS problem arises in efficiently implementing a state-of-the-art LSH solution, called multi-probe random-walk LSH (MP-RW-LSH) [24], for ANNS in Manhattan (L_1) distance. As explained in [24], to compute the value of a random-walk LSH (RW-LSH) function acting on a query vector (as its argument), we need to map an (arbitrarily) given nonnegative even integer ϕ (which can be a very large number) to a ϕ-step random walk. This computation is precisely an ERS problem with X being a single-step random walk. The aforementioned EH3 scheme [5] does not work for this ERS problem for the following reason. It was shown in [24] that, for MP-RW-LSH to work properly, the probability distribution of any computed range-sum $S[a, b)$ must be either identical or close to that of a $(b-a)$-step random walk. This requirement, however, is not generally satisfied by EH3, which destroys all independence beyond 4-wise. In contrast, according to Theorem 9 (in Section 3), our Random Walk (RW)-DST solution strictly satisfies this requirement when it is implemented using 2-wise independent hash functions.

In this work, we make two major and nontrivial contributions. First, we propose a dyadic simulation framework and develop three novel and computationally efficient ERS solutions, namely Gaussian-DST, Cauchy-DST and RW-DST, based on it. Second, we establish a novel

k-wise independence theory that allows our ERS solutions to have both strong provable independence guarantees and low computational complexities.

2 Dyadic Simulation Theory

In this section, we first describe how to generate an arbitrary dyadic range-sum using a *dyadic simulation tree* (DST) of binary splits. After that, we describe three aforementioned DST-based *efficient range-summability* (ERS) solutions for three different target distributions. These three solutions, called Gaussian-DST, Cauchy-DST, and RW-DST (RW for random walk) respectively, follow a common framework and differ only in the binary split procedure. In the rest of the paper, whenever possible, we focus on the design and the efficient implementation of only a single instance of DST. A real-world application usually needs to use many DST instances [8, 7, 24]. These DST instances are independent in the sense that the full vectors of underlying RVs $X_0, X_1, \cdots, X_{U-1}$ generated by them are independent.

Before we describe the dyadic simulation approach, we state the precise problem statement of ERS, which consists of three requirements. First, the underlying RVs $X_0, X_1, \cdots, X_{U-1}$ are i.i.d. with distribution X. Second, every range-sum $S[a, b)$ is equal to $X_a + X_{a+1} + \cdots + X_{b-1}$. Third, given any range $[a, b)$, its range-sum $S[a, b)$ can be computed in $O(\text{polylog}(b - a))$ time. Whereas the second and the third requirements are straightforward to satisfy, to *provably* satisfy the strict independence part of the first requirement, we have to make an idealized assumption that we will elaborate on in Subsection 2.1.

As mentioned earlier, each range in $[0, U)$ can be partitioned into disjoint dyadic ranges. Such a partitioning can usually be done in multiple ways, but only one such way results in the minimum number of partitions. This minimum partitioning is called the *dyadic cover*, which contains at most $O(\log U)$ dyadic ranges [17]. For example, the dyadic cover of $[4, 11)$ contains three dyadic ranges: $[4, 8)$, $[8, 10)$, and $[10, 11)$. In the rest of the paper, we only show how to compute the sum of any dyadic range, since the sum of any general (not necessarily dyadic) range $[a, b)$ can be computed by summing up the dyadic range-sums in the dyadic cover of $[a, b)$. Also as explained earlier, for notational convenience and ease of presentation, we assume that the universe range U is a power of 2.

2.1 Dyadic Simulation Framework

In this section, we describe the dyadic simulation framework, and prove that a DST-based ERS solution satisfies all three requirements specified earlier. We illustrate a DST using a "small universe" example (with $U = 16$) shown in Figure 1. Sitting at the root of the tree is the $S[0, 16)$, which has distribution X^{*16} by initialization. Its two children are the two half-range-sums $S[0, 8)$ and $S[8, 16)$ resulting from splitting $S[0, 16)$, its four grandchildren are the four quarter-range-sums $S[0, 4)$, $S[4, 8)$, $S[8, 12)$ and $S[12, 16)$ resulting from splitting $S[0, 8)$ and $S[8, 16)$ respectively, and so on. At the bottom of the tree are the 16 underlying RVs X_0, X_1, $\cdots$, X_{15}.

Under this model, every dyadic range-sum, including every underlying RV, corresponds to a node in this tree and its value is generated by binary-splitting all its ancestors. The computational complexity of generating a dyadic range-sum is clearly $O(\log U)$ splits. Corollary 1 states the aforementioned surprising result that the computational complexity of generating the sum of any general range is also $O(\log U)$ splits. Hence a DST-based solution satisfies the third requirement above. The remark after the proof of Lemma 10 gives an informal proof of Corollary 1. In addition, under this dyadic simulation framework, the dyadic range-sum at each non-leaf tree node is the sum of two dyadic range-sums at its two

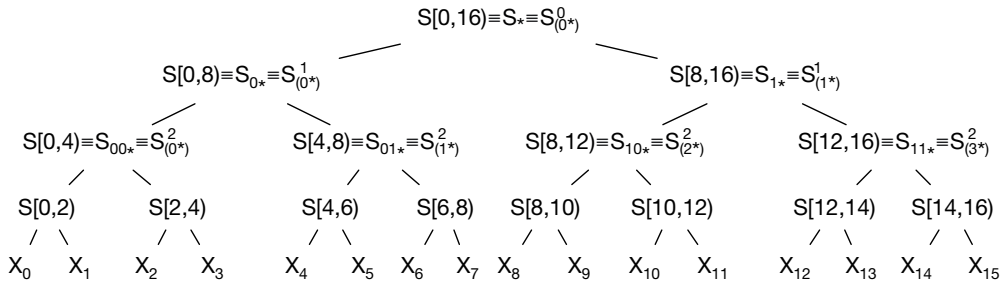


Figure 1 An illustration of the DST.

children. As a result, every dyadic range-sum $S[a,b)$ computed this way is indeed equal to $X_a + X_{a+1} + \cdots + X_{b-1}$. Hence the second requirement above is satisfied.

▶ **Corollary 1.** *For any range $[a,b) \subseteq [0,U)$, the range-sum $S[a,b)$ can be computed in no more than $2\log_2 U$ splits.*

We now introduce the concept of *prefix* that will simplify our presentation next. Viewing the DST as a binary trie, we can index each tree node as a prefix. For example, in Figure 1, the range $[4,8)$ is equivalent to the prefix $01*$ since it contains four binary numbers $4 = (0100)_2$, $5 = (0101)_2$, $6 = (0110)_2$, $7 = (0111)_2$ that share the common prefix $01*$.

Next, we will prove that our DST-based approach satisfies the first requirement (underlying RVs being i.i.d.) above if the split procedure possesses two properties that we call (I) and (II). Suppose a dyadic range-sum $S_{\alpha*}$ that has distribution X^{*2n} is split into $S_{\alpha 0*} + S_{\alpha 1*}$. *Property (I)* is that $S_{\alpha 0*}$ and $S_{\alpha 1*}$ are i.i.d. with distribution X^{*n}. *Property (II)* is that the random vector $\langle S_{\alpha 0*}, S_{\alpha 1*}\rangle$ is a (vector) function of only $S_{\alpha*}$ as far as independence analysis is concerned.

Now, we describe the binary split procedure. To split any $S_{\alpha*}$, we simply generate an RV $L_{\alpha*}$ using a conditional distribution that we will specify next, and then let $S_{\alpha 0*} \triangleq L_{\alpha*}$ and $S_{\alpha 1*} \triangleq S_{\alpha*} - L_{\alpha*}$. Since the split procedure is the same for any $\alpha*$, we drop the subscript $\alpha*$ from $S_{\alpha*}$ and $L_{\alpha*}$ in describing it whenever possible. In the following derivations and proofs, we assume that S is a continuous RV, so its probability density function (pdf) is used; if S is instead a discrete RV, we can use its probability mass function (pmf) instead. To split *S for the first time, a fresh RV L* is generated according to the following conditional pdf:

$$f_{L|S}(L = x|S = z) \triangleq \rho_n(x)\rho_n(z-x)/\rho_{2n}(z), \tag{1}$$

where $\rho_n(\cdot)$ and $\rho_{2n}(\cdot)$ are the pdfs of X^{*n} and X^{*2n} respectively. For notational simplicity, we drop the subscript $L|S$ from $f_{L|S}$ in the sequel. The following theorem states that this split procedure satisfies the aforementioned property (I).

▶ **Theorem 2.** *If S has distribution X^{*2n}, then the conditional distribution of $L|S$ in Equation 1 implies that L and $S - L$ are i.i.d. RVs having distribution X^{*n}.*

Proof. We first calculate the joint pdf of L and $S - L$ as follows

$$f(L = x, S - L = v) = f(L = x|S = x + v)f(S = x + v) = \rho_n(x)\rho_n(v), \tag{2}$$

where Equation 2 can be derived from Equation 1 by replacing z with $x + v$.

Hence we have $f(L = x) = \int_{-\infty}^{\infty} \rho_n(x)\rho_n(v)\mathrm{d}v = \rho_n(x)$. Similarly, $f(S - L = v) = \rho_n(v)$. Hence we have $f(L = x, S - L = v) = f(L = x)f(S - L = v)$, which proves the independence. ◀

We now put the index subscript $\alpha*$ back into S and L, since we need to state results concerning a set of S- and L-terms with different indices. We pause to clarify the mathematical meanings of two emphasized phrases used in stating the split procedure. The first phrase is "for the first time". It means that, in case $S_{\alpha*}$ is to be split again, the same $L_{\alpha*}$, that was generated and used for the first time, must be used again. This is a basic requirement for generating RVs, because the values of RVs should be fixed upon generation. The second phrase is "a fresh RV". It means that each $L_{\alpha*}$ is generated based on only the value of $S_{\alpha*}$ using fresh randomness. As a result, the random vector $\langle S_{\alpha 0*}, S_{\alpha 1*}\rangle$ is a vector function of only $S_{\alpha*}$ as far as independence analysis is concerned, which is precisely property (II). The language of property (II), such as "fresh randomness", is a bit vague right now. It will be further simplified and clarified in Subsection 2.2.

The aforementioned idealized assumption is simply that we can somehow remember the fresh randomness involved in generating each $L_{\alpha*}$ (for the first time), so that property (II) can be ensured. However, since the number of non-leaf prefixes in each DST is $O(U)$, it is typically prohibitively expensive in terms of storage cost to remember such fresh randomness for every $L_{\alpha*}$ generated, and this idealized assumption is impractical. Since property (II) depends on this assumption, it is also impractical. In Section 3, we will introduce a slightly weakened property (II*) that does not require this assumption, yet can still lead to strong provable statistical guarantees.

Before we state and prove the following theorem, we introduce a third notation $S^l_{(i*)}$ for a dyadic range-sum (besides $S[a,b)$ and $S_{\alpha*}$). $S^l_{(i*)}$ represents the same dyadic range-sum as $S_{\alpha*}$, if the number i, written as an l-bit binary number, is (the binary prefix) α. For example, in the example shown in Figure 1, $S^2_{(1*)}$ is equivalent to S_{01*} and $S[4,8)$. Similarly, we denote the L-term involved in splitting $S^l_{(i*)}$ as $L^l_{(i*)}$. Note that if $S_{\alpha*}$ is the same as $S^l_{(i*)}$, then $S_{\alpha 0*}$ and $S_{\alpha 1*}$, the two children of $S_{\alpha*}$, are the same as $S^{l+1}_{((2i)*)}$ and $S^{l+1}_{((2i+1)*)}$ respectively.

Since the DST is a full binary tree, there are 2^l nodes at the l^{th} level down the root. Under this $S^l_{(i*)}$ notation, these 2^l nodes are $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l *)}$, where $\lambda_l = 2^l - 1$ (defined for any l). The following theorem states that for any $1 \le l \le \log_2 U$, these 2^l dyadic range-sums are i.i.d. RVs.

▶ **Theorem 3.** *Suppose that the split procedure satisfies properties (I) and (II). Then, for any l such that $1 \le l \le \log_2 U$, the 2^l dyadic range-sums $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l *)}$ at level l have i.i.d. distribution $X^{*(U/2^l)}$.*

Proof. We prove by induction on l. For the base case when $l = 1$, there are two dyadic range-sums at the 1^{st} level: $S^1_{(0*)}$ and $S^1_{(1*)}$. Since they result from splitting S_*, which has distribution X^{*U} (by initialization), $S^1_{(0*)}$ and $S^1_{(1*)}$ are i.i.d. RVs with distribution $X^{*(U/2)}$ according to property (I).

Now, we prove the case of $l+1$ from that of l. By the induction assumption, for any i, the parent $S^l_{(i*)}$ follows $X^{*(U/2^l)}$, so by property (I), its two children $S^{l+1}_{((2i)*)}$ and $S^{l+1}_{((2i+1)*)}$ are independent and each has the marginal distribution $X^{*(U/2^{l+1})}$. We denote this as *fact (*)*. It remains to show $S^{l+1}_{(0*)}$, $S^{l+1}_{(1*)}$, $\cdots$, $S^{l+1}_{(\lambda_{l+1}*)}$, the generated range-sums on level $l+1$, are independent. By induction assumption, $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l *)}$ are independent. Each $\langle S^{l+1}_{((2i)*)}, S^{l+1}_{((2i+1)*)}\rangle$ is a (vector) function of only $S^l_{(i*)}$, which we called property (II) earlier. Hence the random vectors $\langle S^{l+1}_{((2i)*)}, S^{l+1}_{((2i+1)*)}\rangle$ are independent for different i, which we denote as *fact (**)*.

Therefore, the independence of all values on level $l+1$ follows from the following factorization of the joint cdf for any sequence of values $x_0, x_1, \cdots, x_{\lambda_{l+1}} \in \mathbb{R}$,
$\Pr\left(S^{l+1}_{(0*)} \le x_0, S^{l+1}_{(1*)} \le x_1, \cdots, S^{l+1}_{(\lambda_{l+1}*)} \le x_{\lambda_{l+1}}\right)$
$= \prod_{i=0}^{\lambda_l} \Pr\left(S^{l+1}_{((2i)*)} \le x_{2i}, S^{l+1}_{((2i+1)*)} \le x_{2i+1}\right)$
$= \prod_{i=0}^{\lambda_l} \Pr\left(S^{l+1}_{((2i)*)} \le x_{2i}\right) \Pr\left(S^{l+1}_{((2i+1)*)} \le x_{2i+1}\right) = \prod_{i=0}^{\lambda_{l+1}} \Pr\left(S^{l+1}_{(i*)} \le x_i\right)$, where the first equation is due to fact (**) above and the second is due to fact (*) above. ◀

▶ **Corollary 4.** *The underlying RVs $X_0, X_1, \cdots, X_{U-1}$, which are $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l*)}$ at level $l = \log_2 U$, have i.i.d. distribution X.*

▶ Remark. The following observation, which is a part of fact (*) in the proof of Theorem 3 above, continues to hold when property (II) is taken away, since the proof of this part only needs property (I).

▶ **Observation 5.** *Even if the split procedure satisfies only property (I), each $S^l_{(i*)}$ still has marginal distribution $X^{*(U/2^l)}$.*

The logic of the induction step in the proof of Theorem 3 can be stated as the following lemma, which will be used in the proofs in Section 3.

▶ **Lemma 6.** *If a set of $k > 1$ distinct dyadic range-sums $S^l_{(i_1*)}$, $S^l_{(i_2*)}, \cdots, S^l_{(i_k*)}$ at level l are independent and they are* split conditionally independently*, then their $2k$ children $S^{l+1}_{((2i_1)*)}$, $S^{l+1}_{((2i_1+1)*)}$, $S^{l+1}_{((2i_2)*)}$, $S^{l+1}_{((2i_2+1)*)}$, $\cdots$, $S^{l+1}_{((2i_k)*)}$, $S^{l+1}_{((2i_k+1)*)}$ at level $l+1$ are also independent.*

▶ Remark. Here, "split conditionally independently" means the following two conditions that together lead to fact (**). First, the L-terms involved in these splits, namely $L^l_{(i_1*)}$, $L^l_{(i_2*)}, \cdots, L^l_{(i_k*)}$ are conditionally independent given $S^l_{(i_1*)}$, $S^l_{(i_2*)}, \cdots, S^l_{(i_k*)}$. Second, each such $L^l_{(i*)}$ involved is a (random) function of $S^l_{(i*)}$ only.

2.2 Efficient Range-Summable (ERS) Solutions

As explained earlier, every DST-based solution boils down to generating $L_{\alpha*}$ according to the conditional distribution $f(L_{\alpha*}|S_{\alpha*})$ specified in Equation 1. Although Equation 1 applies to any distribution X in principle, for such a solution to work, two hurdles have to be overcome. The first hurdle is a mathematical one: Nice closed-form formulae for $\rho_n(x)$ (pdf of X^{*n}) and $\rho_{2n}(x)$ (pdf of X^{*2n}), and hence for $f(L_{\alpha*}|S_{\alpha*})$, appear to exist for only a few such X's. For other target distributions, designing DST-based ERS solutions appears to be challenging.

Even when the distribution X is nice so that we have a closed-form formula, we are still facing the second hurdle, which is to generate $L_{\alpha*}$ in a computationally efficient manner. A computational procedure for generating $L_{\alpha*}$ is typically a two-step process as follows. First, we generate a *fresh* (i.e., independent of all other RVs including $S_{\alpha*}$) uniform random μ-bit-long binary string $C_{\alpha*}$ that, if viewed as a nonnegative integer, is uniformly distributed in the set $\{0, 1, 2, \cdots, 2^\mu - 1\}$. Usually $\mu = 32$ provides enough statistical precision. Second, $L_{\alpha*}$ is set to $\theta(C_{\alpha*}, S_{\alpha*})$, where $\theta(x, z)$ is a *deterministic* function designed in such a way that the resulting $L_{\alpha*}$ has the right conditional distribution as specified in Equation 1.

Now we are ready to simplify the language of property (II) as promised earlier. The simplified property (II) is that each $C_{\alpha*}$ is a fresh RV (that is independent of any other RV). As a result, each $L_{\alpha*} \triangleq \theta(C_{\alpha*}, S_{\alpha*})$ is a *fresh* RV that is a function only of $S_{\alpha*}$, which is

precisely property (II). With this simplified property (II), the idealized assumption becomes that each such $C_{\alpha*}$ (not $L_{\alpha*}$) needs to be remembered after it is first generated.

In probability theory, the standard textbook technique, called *inverse transform method* [16], is to let $\theta(x,z) = F^{-1}(x|z)$ where $F(x|z) \triangleq \int_{-\infty}^{x} f(v|z)\mathrm{d}v$ is the conditional cdf of $L_{\alpha*}|S_{\alpha*}$. However, inverse transform is usually not computationally efficient, since the inverse function of the conditional cdf $F^{-1}(x|z)$ usually does not have a closed form, as we will elaborate in Subsection 2.4. We will show that, for all three ERS solutions, we propose alternative designs of $\theta(x,z)$ that are much more efficient, in terms of computational and/or space complexity, than the respective inverse transforms. Finally, when X is a discrete RV (e.g., when X is a single-step random walk), it is possible to precompute $F^{-1}(x|z)$ for all possible values of x and z, and store the values in a table. This technique, called the tabular inverse transform [9], can only be used when the memory cost of storing the table is manageable.

2.3 Gaussian-DST

For notational simplicity, we again drop the subscript $\alpha*$ from $S_{\alpha*}$, $L_{\alpha*}$, and $C_{\alpha*}$ in describing the binary split procedures in the sequel. When X is standard Gaussian $\mathcal{N}(0,1)$, X^{*n} is $\mathcal{N}(0,n)$ with pdf $\rho_n(x) = 1/\sqrt{2\pi n}\cdot\exp(-x^2/(2n))$, and X^{*2n} is $\mathcal{N}(0,2n)$ with pdf $\rho_{2n}(x) = 1/\sqrt{4\pi n}\cdot\exp(-x^2/(4n))$. According to Equation 1, we have $f(L=x|S=z) = \rho_n(x)\rho_n(z-x)/\rho_{2n}(z) = 1/\sqrt{\pi n}\cdot\exp(-(x-z/2)^2/n)$, which can be written as the pdf of $\mathcal{N}(z/2,n/2)$. We generate L from the distribution $\mathcal{N}(z/2,n/2)$ according to (the value of) the random string C as follows. L is set to $z/2+Y$, where Y is a fresh Gaussian RV with distribution $\mathcal{N}(0,n/2)$ generated from C using efficient techniques such as Box-Muller transform [16]. In [7], no specific technique was suggested for generating this L.

2.4 Cauchy-DST

Now we describe how to generate L from C when the target distribution X is standard Cauchy (Cauchy$(0,1)$). By the stability property of Cauchy distribution [8], the n^{th} convolution power X^{*n} is Cauchy$(0,n)$, which has pdf $\rho_n(x) = \left(\pi n\left[1+(x/n)^2\right]\right)^{-1}$. The pdf of X^{*2n} is $\rho_{2n}(x) = \left(2\pi n\left[1+(x/2n)^2\right]\right)^{-1}$. Therefore, by Equation 1, the conditional pdf is

$$f(L=x|S=z) = \frac{\rho_n(x)\rho_n(z-x)}{\rho_{2n}(z)} = \frac{n}{2\pi}\cdot\frac{z^2+4n^2}{(n^2+x^2)(n^2+(z-x)^2)}. \tag{3}$$

In [7], it was suggested that the inverse transform method described above be used to generate L. The rationale offered in [7] was that since the conditional pdf $f(x|z)$ in Equation 3 is a rational fraction, the conditional cdf $F(x|z)$ has a closed-form expression [19], which makes its inverse $F^{-1}(x|z)$ numerically calculable. However, the procedure for calculating $F^{-1}(x|z)$ has a high computational complexity in practice, since the (closed-form) formula of $F(x|z)$ is very complicated.

We propose a much more efficient way of sampling L from $f(x|z)$ based on a Monte Carlo simulation technique called *rejection sampling* [4]. The idea of rejection sampling is that, we instead sample another RV Y from another pdf $\psi(x|z)$ that is computationally easier to sample from than $f(x|z)$. Supposing the value of this sample is x. Then this sample is accepted with probability $\gamma = f(x|z)/(Q\psi(x|z))$ and rejected with probability $1-\gamma$. The rejection sampling step is repeated until a sample of Y is accepted, and the finally accepted sample is (the realized value of) L. Here, this constant Q should be set such that $\gamma \le 1$ for all values of x and z, or in other words $Q \ge \max_{x,z} f(x|z)/\psi(x|z)$. In statistics, a key

objective as well as challenge in designing a rejection sampling procedure is to select $\psi(x|z)$ so that $\max_{x,z} f(x|z)/\psi(x|z)$ and hence this Q can be made as small as possible. Hence the *probability of acceptance*, defined as probability that any sample thus generated is accepted, (which is equal to $1/Q$ as shown in [4] pp. 51) is made as large as possible.

We propose to sample RV Y (whose conditional pdf is $\psi(x|z)$) from the following mixture distribution: Y is equal to Y' or $Y'+z$ each with probability $1/2$ (depending on the value of C), where Y' is a fresh RV with distribution Cauchy$(0,n)$. This Y' can generated from C via the aforementioned inverse transform $Y' = F^{-1}_{Y'}(C) = n\tan(\pi(C-1/2))$; note that, unlike the conditional inverse cdf $F^{-1}(x|z)$ described above, the inverse function of the unconditional cdf $F^{-1}_{Y'}(C)$ here takes a much simpler form and hence can be computed efficiently. It can be shown that the conditional pdf of Y is

$$\psi(L=x|S=z) = \frac{\rho_n(x)+\rho_n(x-z)}{2} = \frac{n}{2\pi}\cdot\frac{2n^2+x^2+(z-x)^2}{(n^2+x^2)(n^2+(z-x)^2)}.$$

We set the parameter Q to 2 so that the probability of acceptance is $1/2$, since for any x and z, we have

$$\frac{f(L=x|S=z)}{\psi(L=x|S=z)} = \frac{4n^2+z^2}{2n^2+x^2+(z-x)^2} = \frac{4n^2+z^2}{2n^2+z^2/2+2(x-z/2)^2} \le 2. \tag{4}$$

.

2.5 Random Walk (RW)-DST

We now describe how to generate L from $S=z$ and C when the target distribution X is a single-step random walk. We first derive the conditional pmf $f(L=x|S=z)$. Since X^{*n} has pmf $\rho_n(x) = 2^{-n}\binom{n}{(n-x)/2}$, and X^{*2n} has pmf $\rho_{2n}(x) = 2^{-2n}\binom{2n}{(2n-x)/2}$, by Equation 1, the conditional pmf is

$$f(L=x|S=z) = \frac{\rho_n(x)\rho_n(z-x)}{\rho_{2n}(z)} = \binom{n}{(n-x)/2}\binom{n}{(n-z+x)/2}\Big/\binom{2n}{n-z/2}, \tag{5}$$

if z is an even integer such that $-2n \le z \le 2n$, x is an integer such that $-n \le x \le n$ and $-n+z \le x \le n+z$, and $n-x$ is even; otherwise $f(L=x|S=z)=0$.

We now introduce a concept that will become handy in the rest of this section. We say that y is a *probable value* of a discrete RV Y, if the probability $P(Y=y)$ is not 0 or vanishingly small. This concept is important here, because we will trade memory space for computation time by precomputing and storing some conditional probability values, and the memory cost could be greatly reduced if we store only those for probable values of S and L conditioned upon S. Now we analyze the asymptotic number of probable values of S and L when n is a large number. For S, only integers that are no larger than $O(\sqrt{n})$ are probable, since as will be shown in the proof of Proposition 7, its pmf $\rho_{2n}(x)$ (on integer values of x) is close to the pdf of $\mathcal{N}(0,2n)$, which is $1/\sqrt{4\pi n}\cdot\exp(-x^2/(4n))$ as shown above. The above formula is not vanishingly small only when $x = O(\sqrt{n})$. Hence, by storing probability values only for the probable values of S, the space complexity reduces from $O(n)$ to $O(\sqrt{n})$. The same can be said about L for a similar reason.

We have tried the aforementioned tabular inverse transform method [9] on $f(L=x|S=z)$. However, even when the probable value trick is used, the memory cost is still very high for most applications. The total memory cost is $O(U)$, since for each of the $\log U$ values of n, we need to store the values of $f(L=x|S=z)$ for all combinations of $O(\sqrt{n})$ probable z values

and $O(\sqrt{n})$ probable x values, and the largest n value is U. For example, when $U = 2^{20}$, the total size of the precomputed tables would still be several gigabytes.

We propose a rejection sampling technique that, in combination with the tabular inverse transform and the probable value trick, provides a fast, space-efficient, and accurate solution to this ERS problem. Like in Subsection 2.4, the rejection sampling method is specified by the RV Y whose conditional pdf (given $S = z$) is $\psi(x|z) = 2^{-n}\binom{n}{(n-x+2\lceil z/4\rceil)/2}$, and the constant Q (defined later). Y can be generated as $Y' + 2\lceil z/4\rceil$, where Y' is a fresh RV with distribution X^{*n} generated from C by tabular inverse transform [9]. Our next step is to determine Q, which is an upper bound on the ratio $f(x|z)/\psi(x|z)$ for each n value and for all probable x and z values (those that are $O(\sqrt{n})$ as explained earlier). For all $n \geq 256$, we know from calculations and from Proposition 7 that this ratio is at most 1.47. Hence, we set $Q = 1.47$ so that the probability of acceptance is $1/1.47 = 0.68$. The rejection sampling operation is computationally efficient, because both $f(x|z)$ and $\psi(x|z)$ can be computed in $O(1)$ time if the factorials $i!$ and $(n-i)!$ are precomputed for probable i values (that is $i = O(\sqrt{n})$). When $n \geq 256$, we use rejection sampling (with $Q = 1.47$). When $n \leq 128$, we use the tabular inverse transform (with the probable value trick) since the table size grows as $O(n)$ as explained earlier. When $U = 2^{20}$ like in the example above, the total size of the precomputed tables (for all 20 values of n) is only several megabytes.

▶ **Proposition 7.** *When n is large and $z = O(\sqrt{n})$ is a probable value, the maximum ratio $\max_{x=O(\sqrt{n})} f(x|z)/\psi(x|z)$ is close to $\sqrt{2} \approx 1.414$.*

Proof. By de Moivre-Laplace Theorem [13], when n is large and $x = O(\sqrt{n})$ is a probable value, $\rho_n(x)$ is close to the pdf of $\mathcal{N}(0, n)$ at x, which is $1/\sqrt{2\pi n} \cdot \exp(-x^2/(2n))$. Similarly, $\rho_n(z-x)$ is close to $1/\sqrt{2\pi n} \cdot \exp(-(z-x)^2/(2n))$, and $\rho_{2n}(x)$ is close to $1/\sqrt{4\pi n} \cdot \exp(-x^2/(4n))$. By straightforward computation, $f(x|z)$ in Equation 5 is close to $1/\sqrt{\pi n} \cdot \exp(-(x - z/2)^2/n)$. Similarly, $\psi(x|z)$, the conditional pdf of Y is close to $1/\sqrt{2\pi n} \cdot \exp(-(x - 2\lceil z/4\rceil)^2/(2n))$. If z is a multiple of 4, the maximum ratio is achieved at $x = z/2$, and the ratio is $\sqrt{2}$. Otherwise, z is an even number but not a multiple of 4, the maximum ratio is achieved at $x = 2\lfloor z/4\rfloor$, and the ratio is $\sqrt{2}\exp(1/n)$, which is close to $\sqrt{2}$ when n is large. ◀

2.6 Speed of Dyadic Simulation

Recall that our idealized and impractical assumption is that we can somehow remember the value of every $C_{\alpha *}$ after it was first generated, with which we can rigorously prove that $X_0, X_1, \cdots, X_{U-1}$ are i.i.d. As mentioned in Subsection 1.2, this assumption can be removed by instead computing each such $C_{\alpha *}$ as $h(\alpha)$, where $h(\cdot)$ is a hash function.

We have implemented the DST framework using an off-the-shelf hash function family called `wyhash` [25]. `wyhash` offers two attractive advantages. First, computationally `wyhash` is very efficient: It takes roughly two nanoseconds for `wyhash` to compute a hash value [25]. Second, it guarantees excellent empirical independence among the values of $(C_{\alpha *})$'s generated in the sense that it passes a number of quality tests in SMhasher, a well-established benchmark for hash functions [23]. To further improve this empirical independence, we use a different (independent) hash function at each level of the DST. The storage cost of a DST is tiny, since each hash function uses only a 32-bit random seed that needs to be remembered. Table 1 shows the average amount of time it takes for a DST to split a Gaussian, Cauchy, and random walk RV respectively, measured on a workstation running Ubuntu 18.04 with Intel(R) Core(TM) i9-10980XE 3.00 GHz CPU. It is a few times faster to split a Gaussian than to

split the other two, largely because the other two involve rejection sampling, which is a relatively computationally intensive process.

As shown in Corollary 1, the generation of any range-sum involves at most $2\log_2 U$ splits, so for typical universe sizes (say $U = 2^{32}$), the total time for generating a range-sum is around or less than 1 µs for all three target distributions in Table 1.

Table 1 Average split time of an RV.

Distribution	Gaussian	Cauchy	Random Walk
Time per split (ns)	4.8	24.8	21.2

3 k-wise Independence Theory for DST

At the end of the previous section, we have shown that, by using a per-level `wyhash` function to hash a prefix $\alpha*$ into a uniformly random string $C_{\alpha*}$, our solutions have high performance and the underlying RVs are empirically independent. However, `wyhash` does not provide any theoretical guarantee concerning independence. In this section, we describe our novel k-wise independence theory that provides both high computational efficiency and strong provable independence guarantees.

Our k-wise independence theory guarantees that the U underlying RVs X_0, $X_1, \cdots, X_{U-1}$ generated by the DST are k-wise independent in the sense that given an arbitrary set of k different indices $i_1, i_2, \cdots, i_k$ in the universe $[0, U)$, the RVs $X_{i_1}, X_{i_2}, \cdots, X_{i_k}$ are independent. To this end, our idea is to use $\log_2 U$ k-wise independent hash functions (instead of `wyhash`) to generate $(C_{\alpha*})$'s. A k-wise independent hash function $h(\cdot)$ has the following property: Given an arbitrary set of k different keys $i_1, i_2, \cdots, i_k$, their hash values $h(i_1), h(i_2), \cdots, h(i_k)$ are independent. Such hash functions are very computationally efficient when k is a small number such as $k = 2$ (roughly 2 nanoseconds per hash just like `wyhash`) and $k = 4$ (several nanoseconds per hash) [3, 21, 15].

Our scheme uses $\log_2 U$ independent k-wise independent hash functions that we denote as $h^l(\cdot)$, for $l = 0, 1, \cdots, \log_2 U - 1$. During the initialization phase, we seed these $\log_2 U$ hash functions each using a uniformly random binary string; once seeded, they are deterministic (hash) functions thereafter. Our scheme can be stated in *literally one sentence:* Each such (seeded and fixed) $h^l(\cdot)$ is solely responsible for hash-generating all random strings $C_{\alpha*}$ in which the binary prefix α is a l-bit number. For example, when a random string $C_{\alpha*}$ is needed for computing a range-sum, we rewrite $C_{\alpha*}$ into $C^l_{(i*)}$ in the same way as we did on $S_{\alpha*}$ in the second last paragraph before Theorem 3. Then, $C^l_{(i*)}$ is hash-generated as $h^l(i)$ just like using `wyhash`. Note that the hash function $h^l(\cdot)$ is a random function before it is seeded, so $h^l(i)$ for any i can be viewed as a RV where the randomness comes from the seed of $h^l(\cdot)$. We will use this view in the proof that follows.

The construction above weakens property (II) slightly. The weakened one, called *property (II*)*, is that, at any level l, any k distinct range-sums $S^l_{(i_1*)}, S^l_{(i_2*)}, \cdots, S^l_{(i_k*)}$ are split conditionally independently. The construction above can guarantee property (II*), because their "split seeds" $C^l_{(i_1*)}, C^l_{(i_2*)}, \cdots, C^l_{(i_k*)}$ are not only independent among themselves (thanks to $h^l(\cdot)$ being k-wise independent) but also independent of $S^l_{(i_1*)}, S^l_{(i_2*)}, \cdots, S^l_{(i_k*)}$ (since $h^l(\cdot)$ is a fresh hash function that has never been used in hash-generating any such $S^l_{(i*)}$). With this construction, the DST has the following nice k-wise independence property at every level.

▶ **Theorem 8.** *If every $h^l(\cdot)$, for $1 \le l < \log_2 U$, is k-wise independent, then for any l such that $1 \le l \le \log_2 U$, the 2^l range-sums $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l *)}$ are k-wise independent.*

Proof. The proof is similar to that of Theorem 3 by induction. For the base case when $l = 1$, there are two dyadic range-sums at the 1^{st} level: $S^1_{(0*)}$ and $S^1_{(1*)}$. Since they result from splitting S_*, which follows X^{*U} (by initialization), $S^1_{(0*)}$ and $S^1_{(1*)}$ are i.i.d. RVs according to property (I).

Now, we prove the induction on level $l+1$ from level l. For any fixed set of k indices i_1, i_2, $\cdots$, i_k on level $l+1$, we need to prove $S^{l+1}_{(i_1*)}$, $S^{l+1}_{(i_2*)}$, $\cdots$, $S^{l+1}_{(i_k*)}$ are independent. This follows from Lemma 6, since these k elements are the children of no more than k parents after duplicates are removed. These parents, no more than k in number, are independent by the induction hypothesis and are split conditionally independently by property (II*). ◀

Theorem 8 implies that the underlying RVs $X_0, X_1, \cdots, X_{U-1}$, which are the U singleton range-sums at level $\log_2 U$, are also k-wise independent. This implication, however, is far from capturing the "full theoretical strength" of our k-wise independence theory. For example, under the assumption that every $h^l(\cdot)$, for $1 \le l < \log_2 U$, is 2-wise independent, Theorem 8 can only guarantee that the underlying RVs are 2-wise independent. In contrast, under this assumption, Theorem 9 guarantees a distributional property that is much stronger than the underlying RVs being 2-wise independent. Here we explain this point by an example. Suppose we take out this assumption, and instead make the alternative assumption that the underlying RVs are k-wise independent for a certain k. It is not hard to prove that, to guarantee the same distributional property, we would have to assume that k is as large as U, or in other words that the underlying RVs are all-wise independent.

This distributional property is in practice very useful. As mentioned earlier in Subsection 1.3, when X is a single-step random walk, this distributional property satisfies the requirement of RW-LSH. Hence any RW-LSH value computed using our RW-DST scheme is statistically indistinguishable from the original RW-LSH value (computed in the original inefficient manner), as long as every $h^l(\cdot)$, for $1 \le l < \log_2 U$, is 2-wise independent. Theorem 9 is an immediate corollary of Lemma 10.

▶ **Theorem 9.** *If every $h^l(\cdot)$, for $1 \le l < \log_2 U$, is 2-wise independent, then for any range $[a, b) \subseteq [0, U)$, the range-sum $S[a, b)$ has marginal distribution $X^{*(b-a)}$.*

▶ **Lemma 10.** *If every $h^l(\cdot)$ is 2-wise independent, then for any $1 \le l < \log_2 U$ and any integers a, b such that $0 \le a \le b < 2^l$, the following two properties hold.*

1. *The three RVs $\sum_{i=a+1}^{b-1} S^l_{(i*)}$, $S^l_{(a*)}$, and $S^l_{(b*)}$ are independent.*
2. *The range-sum $S[aU/2^l, (b+1)U/2^l) = \sum_{i=a}^{b} S^l_{(i*)}$ has distribution $X^{*((b-a+1)U/2^l)}$, where $(b-a+1)U/2^l$ is the number of underlying RVs contained in the range $[aU/2^l, (b+1)U/2^l)$.*

Before we start the proof, we note that Observation 5, which states that the marginal distribution of any range-sum $S^l_{(i*)}$ being $X^{*(U/2^l)}$ continues to hold despite the weakening of property (II) in this section.

Proof. The proof for Lemma 10 is by induction on l. For the base case when $l = 1$, the three RVs, after 0's and duplicates are removed, belong to the set of two range-sums on the first level, $S^1_{(0*)}$ and $S^1_{(1*)}$. These two range-sums have distribution i.i.d. $X^{*(U/2)}$ thanks to property (I). This leads to the two properties on the first level.

We now prove the case of level $l+1$ from that of level l. We first define the following notations: $a' \triangleq \lfloor a/2 \rfloor$ (so that $S^l_{(a'*)}$ is the parent of $S^{l+1}_{(a*)}$); $b' \triangleq \lfloor b/2 \rfloor$; $\tilde{i}$ is defined as $i+1$ if i is even (the younger of the siblings) and as $i-1$ otherwise so that $S^l_{(\tilde{i}*)}$ is always the other sibling of $S^l_{(i*)}$. Without loss of generality, we assume $a' < b'$, since otherwise ($a' = b'$) the two induction claims become trivial. The induction claim of the first

property that $\sum_{i=a+1}^{b-1} S^{l+1}_{(i*)}$ and $\langle S^{l+1}_{(a*)}, S^{l+1}_{(b*)}\rangle$ are independent holds, due to the following two facts: (i) $\sum_{i=a'+1}^{b'-1} S^{l}_{(i*)}, S^{l+1}_{(a*)}, S^{l+1}_{(\tilde{a}*)}, S^{l+1}_{(b*)}$, and $S^{l+1}_{(\tilde{b}*)}$ are independent; (ii) $\sum_{i=a+1}^{b-1} S^{l+1}_{(i*)}$ is a deterministic function of $\langle \sum_{i=a'+1}^{b'-1} S^{l}_{(i*)}, S^{l+1}_{(\tilde{a}*)}, S^{l+1}_{(\tilde{b}*)}\rangle$ in the sense that $\sum_{i=a+1}^{b-1} S^{l+1}_{(i*)} = \sum_{i=a'+1}^{b'-1} S^{l}_{(i*)} + S^{l+1}_{(\tilde{a}*)}\mathbf{1}_{even}(a) + S^{l+1}_{(\tilde{b}*)}\mathbf{1}_{odd}(b)$, where $\mathbf{1}_{even}(a)$ is 1 if a is an even integer and is 0 otherwise, and similarly $\mathbf{1}_{odd}(b)$ is 1 if b is an odd integer and is 0 otherwise.

We now prove fact (i). By the induction assumption, the three RVs $\sum_{i=a'+1}^{b'-1} S^{l}_{(i*)}$, $S^{l}_{(a'*)}$, and $S^{l}_{(b'*)}$ are independent. Since $h^l(\cdot)$ is a fresh 2-wise independent hash function, property (II*) holds for $k = 2$, so the four children $S^{l+1}_{(a*)}$, $S^{l+1}_{(\tilde{a}*)}$, $S^{l+1}_{(b*)}$, and $S^{l+1}_{(\tilde{b}*)}$ are independent by Lemma 6. Furthermore, these four children and $\sum_{i=a'+1}^{b'-1} S^{l}_{(i*)}$ together are independent, because by property (II*), $\langle S^{l+1}_{(a*)}, S^{l+1}_{(\tilde{a}*)}\rangle$ is a function of only $S^{l}_{(a'*)}$ but not $\sum_{i=a'+1}^{b'-1} S^{l}_{(i*)}$, which is composed of the other range-sums, and similarly $\langle S^{l+1}_{(b*)}, S^{l+1}_{(\tilde{b}*)}\rangle$ is a function of only $S^{l}_{(b'*)}$. By the induction assumption, $\sum_{i=a'+1}^{b'-1} S^{l}_{(i*)}$ is independent of these four children.

We now prove the induction claim of the second property that $\sum_{i=a}^{b} S^{l+1}_{(i*)}$ has distribution X^{*n} with $n = (b-a+1)U/2^{l+1}$. By the second property in the induction assumption, $\sum_{i=a'+1}^{b'-1} S^{l}_{(i*)}$ has distribution $X^{*n'}$, where $n' = (b'-a'-1)U/2^l$ is the number of $(X_i)'s$ (underlying RVs) contained in its range $[(a'+1)U/2^l, b'U/2^l)$. To see why the induction claim holds, we have to go through four possible cases on the parities of a and b. We show the most inclusive case where a is even and b is odd, and the other three cases are just similar. In this case, $b-a = 2(b'-a')+1$, so $n = (2(b'-a')+2)U/2^{l+1} = (2(b'-a')-2+4)U/2^{l+1} = n' + 4U/2^{l+1}$. $\sum_{i=a}^{b} S^{l+1}_{(i*)}$ has distribution X^{*n}, because it is the sum of the following five independent RVs (*cf.* fact (i)): $\sum_{i=a'+1}^{b'-1} S^{l}_{(i*)}$, $S^{l+1}_{(a*)}$, $S^{l+1}_{(\tilde{a}*)}$, $S^{l+1}_{(b*)}$, and $S^{l+1}_{(\tilde{b}*)}$. The first RV has distribution $X^{*n'}$, and the other four RVs each has distribution $X^{*(U/2^{l+1})}$ by Observation 5. ◀

▶ Remark. In this proof, to compute the range-sum $S[a,b) = \sum_{i=a}^{b-1} S^{\log_2 U}_{(i*)}$, at most two splits need to performed at each level $0 \le l < \log_2 U$, on namely $S^{l}_{(a_l*)}$ and $S^{l}_{(b_l*)}$ (they can be the same node), where $a_l = \lfloor a2^l/U \rfloor$ and $b_l = \lfloor b2^l/U \rfloor$. This implies Corollary 1.

4 Related Work

Since the contribution of this work is a new and practical solution approach to the ERS problem, we focus only on related works on ERS. The ERS problem was first formulated in [5]. In [5], the aforementioned EH3, which is the first ERS solution, was proposed to augment the AMS sketching [1] technique. The EH3-augmented AMS solves a wide range of new data streaming problems, such as estimating the size of spatial joins and the selectivity of histogram buckets, and outperforms previous ad-hoc solutions [17]. In the ERS literature, the one most related to this work is [7]. We have compared our work with [7] in several places throughout this paper.

All existing ERS solutions except [7] are proposed for the case in which the target distribution X is a single-step random walk. Among them, EH3 [5] is the best known and has been compared with our dyadic simulation approach in Subsection 1.3. The "3" in EH3 refers to the fact that underlying RVs generated by EH3 are provably 3-wise independent. BCH3 [17] is another ERS scheme that also guarantees 3-wise independence. Although BCH3 is faster to compute than EH3, the underlying RVs generated by BCH3 are even more

strongly correlated (beyond 4-wise) [17] than those by EH3. RM7 [2], which guarantees 7-wise independence, is the only existing ERS scheme that goes beyond 3-wise, but it is too slow to be practical. Empirically, RM7 takes more than 26 ms to compute a single range-sum [17], whereas for dyadic simulation, the time is typically about 1 µs as shown in Subsection 2.6.

Besides performance, another issue for these schemes is that they destroy all empirical independence beyond 4-wise (in the cases of EH3 and BCH3) and 8-wise (in the case of RM7). For existing ERS solutions except dyadic simulation, this destruction (of empirical independence) is unavoidable due to the fact that they all solve an ERS problem by crafting a "magic" hash function that is based on error correction codes. For example, in RM7 this magic hash function is defined by an instantiation of the Reed-Muller (RM) code. In contrast, in our dyadic simulation approach, the ERS is achieved through a DST that does not require any such magic hash function: For all practical purposes, `wyhash` will do, as explained earlier. This difference allows our approach to generalize to more target distributions and more applications.

A marginally related range-efficient computing problem to ERS, called efficient range minimizability (ERM), has been studied in the contexts of data streaming and computational geometry. In ERM, we would like to efficiently compute the minimum value of the RVs (that each has distribution X) in a range. An example ERM problem is when X is a uniform distribution in the interval $(0, 1)$. We have come up with a new efficient solution to this problem, but cannot include it in this paper in the interest of space. Any efficient solution (including ours) to this problem can be used, in combination with the MinHash sketch [6], to solve the range-efficient F_0 (estimation) problem [14, 20]: to efficiently estimate the number of distinct elements (F_0) in a data stream with range-updates. Existing solutions to this problem, such as range-efficient sampling [14, 20], are sampling-based in the sense they maintain a select subset of sampled data items instead of a sketch (e.g., accumulators like in [8]). The range-efficient F_0 problem has been generalized to high-dimensional spaces, where it is called the Klee's measure problem in computational geometry [22, 18, 10]. Existing solutions to Klee's measure problem are also sampling based.

5 Conclusion

In this work, we propose *dyadic simulation*, a novel solution framework to ERS that extends and improves existing frameworks in a fundamental and systematic way. We develop three novel ERS solutions for Gaussian, Cauchy, and single-step random walk distributions. We also propose novel rejection sampling techniques to make these solutions computationally efficient. Finally, we develop a novel k-wise independence theory of DSTs that provide both high computational efficiency and strong provable independence guarantees.

References

1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237823.

2 A. Robert Calderbank, Anna C. Gilbert, Kirill Levchenko, Shan Muthukrishnan, and Martin Strauss. Improved range-summable random variable construction algorithms. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 840–849, USA, 2005. Society for Industrial and Applied Mathematics. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.6849.

3 J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. doi:10.1016/0022-0000(79)90044-8.

4 George Casella, Christian P. Robert, and Martin T. Wells. *Generalized Accept-Reject Sampling Schemes*, volume Volume 45 of *Lecture Notes–Monograph Series*, pages 342–347. Institute of Mathematical Statistics, Beachwood, Ohio, USA, 2004. doi:10.1214/lnms/1196285403.

5 Joan Feigenbaum, Sampath Kannan, Martin J. Strauss, and Mahesh Viswanathan. An approximate L_1-difference algorithm for massive data streams. *SIAM Journal on Computing*, 32(1):131–151, 2002. doi:10.1137/S0097539799361701.

6 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.

7 Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 389–398, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/509907.509966.

8 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, May 2006. doi:10.1145/1147954.1147955.

9 George Marsaglia, Wai Wan Tsang, and Jingbo Wang. Fast generation of discrete random variables. *Journal of Statistical Software, Articles*, 11(3):1–11, 2004. doi:10.18637/jss.v011.i03.

10 Kuldeep S. Meel, N.V. Vinodchandran, and Sourav Chakraborty. *Estimating the Size of Union of Sets in Streaming Models*, pages 126–137. Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3452021.3458333.

11 S. Muthukrishnan and Martin Strauss. *Approximate Histogram and Wavelet Summaries of Streaming Data*, pages 263–281. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. doi:10.1007/978-3-540-28608-0_13.

12 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. doi:10.1007/BF01305237.

13 Athanasios Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 1984.

14 A. Pavan and Srikanta Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM Journal on Computing*, 37(2):359–379, 2007. doi:10.1137/050643672.

15 Mihai Pundefinedtraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3), June 2012. doi:10.1145/2220357.2220361.

16 Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*, page 43. Springer New York, 2004. doi:10.1007/978-1-4757-4145-2_2.

17 Florin Rusu and Alin Dobra. Pseudo-random number generation for sketch-based estimations. *ACM Trans. Database Syst.*, 32(2):11–es, June 2007. doi:10.1145/1242524.1242528.

18 Gokarna Sharma, Costas Busch, Ramachandran Vaidyanathan, Suresh Rai, and Jerry L. Trahan. Efficient transformations for Klee's measure problem in the streaming model. *Computational Geometry*, 48(9):688–702, 2015. doi:10.1016/j.comgeo.2015.06.007.

19 James Stewart. *Calculus: Early Transcendentals*. Brooks/Cole, 4 edition, 1999.

20 He Sun and Chung Keung Poon. Two improved range-efficient algorithms for F_0 estimation. *Theoretical Computer Science*, 410(11):1073–1080, 2009. Algorithms, Complexity and Models of Computation. doi:10.1016/j.tcs.2008.10.031.

21 Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 615–624, USA, 2004. Society for Industrial and Applied Mathematics.

22 Srikanta Tirthapura and David Woodruff. Rectangle-efficient aggregation in spatial data streams. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles*

of Database Systems, PODS '12, pages 283–294, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2213556.2213595.

23 Reini Urban and et al. Smhasher: Hash function quality and speed tests. GitHub repository, https://github.com/rurban/smhasher. accessed on Jul 23, 2021.

24 Huayi Wang, Jingfan Meng, Long Gong, Jun Xu, and Mitsunori Ogihara. MP-RW-LSH: An efficient multi-probe lsh solution to ANNS-L1. *Proc. VLDB Endow.*, 14(13):3267–3280, September 2021. doi:10.14778/3484224.3484226.

25 Yi Wang. wyhash: The dream fast hash function and random number generators. GitHub repository, https://github.com/wangyi-fudan/wyhash. Accessed on Feb 9, 2021.

Discovering Event Queries from Traces: Laying Foundations for Subsequence-Queries with Wildcards and Gap-Size Constraints

Sarah Kleest-Meißner
Humboldt-Universität zu Berlin, Germany

Rebecca Sattler
Humboldt-Universität zu Berlin, Germany

Markus L. Schmid
Humboldt-Universität zu Berlin, Germany

Nicole Schweikardt
Humboldt-Universität zu Berlin, Germany

Matthias Weidlich
Humboldt-Universität zu Berlin, Germany

Abstract

We introduce subsequence-queries with wildcards and gap-size constraints (swg-queries, for short) as a tool for querying event traces. An swg-query q is given by a string s over an alphabet of variables and types, a global window size w, and a tuple $c = ((c_1^-, c_1^+), (c_2^-, c_2^+), \ldots, (c_{|s|-1}^-, c_{|s|-1}^+))$ of local gap-size constraints over $\mathbb{N} \times (\mathbb{N} \cup \{\infty\})$. The query q matches in a trace t (i. e., a sequence of types) if the variables can uniformly be substituted by types such that the resulting string occurs in t as a subsequence that spans an area of length at most w, and the i^{th} gap of the subsequence (i. e., the distance between the i^{th} and $(i{+}1)^{\text{th}}$ position of the subsequence) has length at least c_i^- and at most c_i^+. We formalise and investigate the task of discovering an swg-query that describes best the traces from a given sample $\mathcal{S}$ of traces, and we present an algorithm solving this task. As a central component, our algorithm repeatedly solves the matching problem (i. e., deciding whether a given query q matches in a given trace t), which is an NP-complete problem (in combined complexity). Hence, the matching problem is of special interest in the context of query discovery, and we therefore subject it to a detailed (parameterised) complexity analysis to identify tractable subclasses, which lead to tractable subclasses of the discovery problem as well. We complement this by a reduction proving that any query discovery algorithm also yields an algorithm for the matching problem. Hence, lower bounds on the complexity of the matching problem directly translate into according lower bounds of the query discovery problem. As a proof of concept, we also implemented a prototype of our algorithm and tested it on real-world data.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Data structures and algorithms for data management; Theory of computation → Problems, reductions and completeness; Information systems → Data mining

Keywords and phrases event queries on traces, pattern queries on strings, learning descriptive queries, complexity of query evaluation and query learning

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.18

Funding *Sarah Kleest-Meißner*: Supported by the German Research Foundation (DFG), CRC 1404: "FONDA: Foundation of Workflows for Large-Scale Scientific Data Analysis".
Rebecca Sattler: Supported by the German Research Foundation (DFG), CRC 1404: "FONDA: Foundation of Workflows for Large-Scale Scientific Data Analysis".
Markus L. Schmid: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 416776735 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 416776735).

1 Introduction

Event stream processing emerged as a computational paradigm that is based on the continuous evaluation of queries over event streams [10, 22]. Respective systems enable the definition of queries that detect patterns of events, i.e., a match of a query is a joint occurrence of events that satisfy certain properties. Common languages for the definition of event queries, as reviewed in [22, 5], include means to specify such properties related to the order of events, their types, as well as a time window for their joint occurrence.

Event stream processing has traditionally been used for reactive applications in diverse domains, reaching from infrastructure monitoring [7] through financial trading [29] to urban transportation [6]. Here, event queries are specified by expert users to detect situations of interest. For example, consider the case of monitoring a large compute cluster based on events that indicate that a particular *job* was *submitted*, *scheduled*, *suspended*, *resumed*, and *updated* while running, before it either *completes* successfully, is *evicted* due to preemption, *fails* with an error, or is *aborted* manually. In addition, events may carry information on the job's *priority* and the *machine* on which the job runs. Given such event streams, abnormal job execution materialises as an event pattern. For instance, a job may be evicted and rescheduled twice on the same machine, with low and unchanged priority, before it eventually fails. Adopting the SASE language [31], Listing 1 formalises this query.

Listing 1 Event query defined in the SASE language [31]. A match is a sequence of events of the respective transition types (`Evict`, `Schedule`, `Fail`), all being related to the same job (with `[job]` being a short-hand for `a.job=b.job`, `a.job=c.job`, etc.) and to the same machine (`[machine]`), while the priority remains unchanged (`b.prio=low`, `d.prio=low`); all within one hour.

```
PATTERN SEQ(Evict a, Schedule b, Evict c, Schedule d, Fail e)
WHERE [job] AND [machine] AND b.prio=low AND d.prio=low
WITHIN 1h
```

In pro-active applications where an event query shall anticipate a situation of interest to prepare for it accordingly [4], users often only know when a specific situation occurred, but they have no or only partial knowledge on the patterns that indicate that the situation will materialise soon. In the above example, a user may be aware that certain jobs fail execution, and potentially possesses anecdotal evidence that only jobs with low priority are affected, but it is not known that repeated eviction and re-scheduling on the same machine may serve to forecast this failure. Against this background, it was suggested to automatically infer the event patterns of interest from historic event data [21, 24], which can then be interpreted and validated by a user, thereby mitigating the risk of overfitting and enabling traceability.

Specifically, historic event data is first split into *traces* that are considered independent observations (e.g., events may be grouped by *jobs* and *machines*), before the traces related to the situation of interest are identified (e.g., based on the presence of *failure* events). See also Figure 1. Those then serve as a *sample* of positive examples for the discovery process of event queries that are (i) *correct* (they match a share of the sample that is above a given support threshold), and (ii) *descriptive* (any query that is more specific is no longer correct).

In this paper, we introduce *subsequence-queries with wildcards and gap-size constraints* (*swg-queries*) as a formal model that covers the essence of the task of event query discovery. Intuitively, an swg-query is a string over an alphabet of variables and types, a global window size and a tuple of local gap-size constraints. It matches a trace, i.e., a sequence of types, if its variables can be substituted by types, such that the resulting string represents a subsequence of the trace, and the global window size and the local gap-size constraints are satisfied.

```
Schedule(time=21, job=3, machine=m5v, prio=low)
Evict(time=42, job=3, machine=m5v)
Schedule(time=52, job=5, machine=m5v, prio=high)
Schedule(time=112, job=3, machine=m5v, prio=low)
Update(time=142, job=5, machine=m5v)
Complete(time=217, job=5, machine=m5v)
Suspend(time=276, job=3, machine=m5v)
```

$t_1 =$ SchLow E SchLow Sus
$t_2 =$ SchHigh U C

(a) Events and their representation as traces, i.e., sequences of types, per job and machine.

$t_1 =$ SchLow E SchLow Sus
$t_2 =$ SchHigh U C
$t_3 =$ SchHigh E SchHigh U E F
$t_4 =$ SchHigh U U C
$t_5 =$ SchLow E SchLow Sus Res E A
$t_6 =$ SchLow U U E SchLow E A

(b) Database of six traces; t_3, t_5, and t_6 denote failed (F) and aborted (A) executions.

Figure 1 Illustration of the setting of event query discovery.

Let us illustrate how the discovery of event queries defined in common languages for event stream processing translates into the discovery of swg-queries.

▶ **Example 1.1.** The seven events of Figure 1a indicate lifecycle transitions of two jobs. Encoding the events by types that model the lifecycle transitions, potentially combined with additional payload data (e.g., `Schedule` events are modelled by types `SchLow` and `SchHigh`, depending on the assigned priority), we derive two traces t_1 and t_2 (the order of types follows from the events' occurrence times). Note that we partitioned the events by the corresponding job such that t_1 and t_2 consists only of events associated to job 3 and 5, respectively. Now assume that in this way we have obtained six traces (Figure 1b), and that t_3, t_5 and t_6 are identified as traces that relate to the situation of interest: abnormal job execution that leads to failure (F) or abortion (A). Thus, $\{t_3, t_5, t_6\}$ is our sample of historic event data for which query discovery is to be performed. Each trace contains a subsequence of a scheduling event of low or high priority (SchLow or SchHigh), an eviction event (E), a scheduling event of the same priority as the initial one (SchLow or SchHigh), and another eviction event (E). These commonalities can be captured by an swg-query by explicitly representing the eviction events by the type E, while the occurrence of the scheduling events is captured by two occurrences of the same variable x, i. e., a suitable swg-query would be $x\mathsf{E}x\mathsf{E}$. Since the semantics of swg-queries is based on subsequences, in between the types and variables of $x\mathsf{E}x\mathsf{E}$ also other types like U, Sus and Res may occur. Furthermore, the query matches the traces with global window size of 6. Note that the global window size w has to be at least 6 to describe t_5 and t_6, because from the first scheduling event to the second eviction event both traces consist of 6 events. As gap-size constraint we could choose any tuple of the form $c = ((0, i), (0, j), (0, k))$, where $i, k \in \mathbb{N}_{\geqslant 2} \cup \{\infty\}$ and $j \in \mathbb{N} \cup \{\infty\}$ (observe that in t_5 two events occur between the last scheduling and eviction event, and in t_6 two events occur between the first scheduling and eviction event; thus, any c with $i, k \geqslant 2$ is suitable here). The most restrictive choice that suitably describes our example is $w = 6$ and $c = ((0, 2), (0, 0), (0, 2))$.

Languages for complex event processing describe queries which correlate events to detect event patterns, using operators such as sequencing, conjunction, Kleene closure, negation and variables which may be bound to events in a stream [5, 10, 31]. However, the differences in semantics and expressiveness of different languages are not yet well-studied and the formalisation of event queries is an active area of research. Defining a query is usually assumed to be a manual step requiring expert knowledge. The iCEP System [24] and the IL-Miner [21] form an exception, but both are limited regarding their expressive power and scalability, among others. Furthermore, the complexity of the event mining problem is still open; it has not yet received an in-depth investigation from a theoretical perspective.

As illustrated in Example 1.1, we want to find situations of interest that occur in many traces of the given sample set of traces. At a first glance this seems to be a similar problem as solved in the context of frequent sequence mining. Algorithms in this research area as [2, 8, 9, 30] require the a-priori definition of event abstractions; in particular, fine-grained conditions to generalise individual events or to correlate several events are not discovered by these approaches.

In this paper, we try to overcome the issues raised above by proposing a basic language that supports sequencing, variables that range over single events, and different kinds of window size constraints, i.e. local gap-size constraints and a global window size. Hence our swg-queries are located at the core of complex event processing languages while based on classical concepts of theoretical computer science: subsequences and (string) patterns with variables. Subsequences have extensively been studied both in a purely combinatorial sense (in formal language theory, logic and combinatorics on words) and algorithmically (in string algorithms and bioinformatics); see the introductions of the recent papers [20, 11] for a comprehensive list of relevant pointers. Patterns with variables are introduced by Angluin [3] and they play a central role for inductive inference, in formal language theory and combinatorics on words (see [28, 23, 25]). These *Angluin-style* patterns are just strings with terminal symbols $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ and variables $x_1, x_2, x_3, \ldots$, e. g., $p = x_1\,\mathsf{a}\,x_1\,\mathsf{b}\,x_2$, and they match exactly the strings that can be obtained by uniformly replacing the variables by some terminal strings (i. e., exactly the strings $u\,\mathsf{a}\,u\,\mathsf{b}\,v$ with $u, v \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots\}^*$ for the example pattern). Syntactically, our swg-queries are Angluin-style patterns, but with the semantics adapted to event streams: variables only range over single symbols; and the query matches if, after replacing the variables by events, it occurs as a subsequence that satisfies the window size and gap-size constraints.

Despite the fundamental semantic differences between swg-queries and Angluin-style patterns, it is possible to adapt concepts and algorithms from inductive inference of the so-called *pattern languages* that can be described by Angluin-style patterns. Most importantly, the classical concept of *descriptive patterns*, already introduced in [3] for Angluin-style patterns (see also [18, 19]), can be adapted to swg-queries. For a given sample of strings, a descriptive Angluin-style pattern can be computed by Shinohara's algorithm [27], which employs a rather simple and robust algorithmic idea (see also [14]). The main result of this paper is that Shinohara's algorithm can also be adapted for computing descriptive swg-queries, and thus to the case of swg-query discovery. Due to the semantic differences, this adaption is non-trivial and requires the development of suitable technical tools. Moreover, our variant of the algorithm has the following special features:

- it allows a *support threshold* (i. e., a lower bound on the number of matched elements of the sample),
- it can be parameterised by any class of queries that satisfies some mild closure properties (this is crucial for dealing with complexity issues, as explained further below),
- it can also be used in order to check whether a given query is descriptive.

The algorithm computes a descriptive query by performing a number of iterations that is bounded by a low-degree polynomial in the length of the query and the size of the sample. However, in each iteration, the algorithm calls a sub-routine that solves the *matching problem*: Decide whether a given query matches a given trace. Just like for Angluin-style patterns, matching for swg-queries is intractable; thus, it constitutes a complexity bottle-neck in our algorithm. However, as another similarity between Angluin-style patterns and our swg-queries (see [14]), the matching problem for swg-queries reduces to the problem of computing descriptive queries. This means that this complexity bottle-neck inevitably exists in any

possible algorithm for computing descriptive swg-queries. Therefore, we perform a thorough complexity analysis of the matching problem for swg-queries (note that for Angluin-style patterns the complexity of the matching problem is well-understood (see [15, 16])). On the positive side, since our algorithm can be parameterised by any arbitrary class of queries (with some natural closure properties), restrictions of swg-queries that lead to a tractable matching problem also lead to tractable computation of descriptive queries via our algorithm.

Typical brute-force approaches to data mining tasks (e. g., the well-known apriori-algorithm [1]) produce the full set of all objects of interest, but operate in exponential running time. Our algorithm produces only one descriptive query, but, for classes of queries with tractable matching problem, is rather fast. Moreover, by considering all possible runs of the algorithm, we can produce a large class of descriptive queries, although not all of them. We further illustrate the feasibility of our approach by developing a prototypical implementation of our algorithm and a performing a preliminary experimental study on a real-world dataset of cluster monitoring traces at Google [26].

The remainder of this paper is structured as follows. In Section 2, we propose swg-queries as an adaption of Angluin-style patterns to event streams, and we develop some technical tools for this query class. In Section 3, we present the algorithm, based on Shinohara's work on discovering descriptive patterns, which can be used (a) for discovering a descriptive swg-query from a given sample, and (b) for checking whether a given swg-query is descriptive with respect to a sample. Motivated by the observation that computing descriptive queries reduces to the matching problem, and vice versa (this connection is discussed at the end of Section 3), we perform in Section 4 a thorough (classical and parameterised) complexity analysis of the matching problem. Our results point out for which classes of queries efficient computation of descriptive queries is possible, and for which classes this is an intractable problem. Finally, in Section 5, we report some experimental results obtained by applying a prototypical implementation to a real-world dataset, and we give some concluding remarks. Due to space restrictions many technical details had to be deferred to the paper's full version.

2 Traces and Queries

By $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{N}_{\geqslant 1}$ we denote the set of integers, non-negative integers, and positive integers, respectively. For $\ell \in \mathbb{N}$ we let $[\ell] = \{i \in \mathbb{N}_{\geqslant 1} \ : \ 1 \leqslant i \leqslant \ell\}$. For a non-empty set A we write A^* (and A^+) for the set of all strings (the set of all non-empty strings) built from symbols in A. By $|s|$ we denote the length of a string s, and for a position $i \in [|s|]$ we write $s[i]$ to denote the letter at position i in s. A *factor* of a string $s \in A^*$ is a string $t \in A^*$ such that s is of the form $s_1 t s_2$ for $s_1, s_2 \in A^*$.

An *embedding* is a mapping $e : [\ell] \to [n]$ with $\ell \leqslant n$ such that $i < j$ implies $e(i) < e(j)$ for all $i, j \in [\ell]$. Let s and t be two strings with $|s| \leqslant |t|$. We say that s is a *subsequence of t with embedding $e : [|s|] \to [|t|]$*, if e is an embedding and $s[i] = t[e(i)]$ for every $i \in [|s|]$. We write $s \preccurlyeq_e t$ to indicate that s is a subsequence of t with embedding e; and we write $s \preccurlyeq t$ to indicate that there exists an embedding e such that $s \preccurlyeq_e t$.

We model traces as finite, non-empty strings over some (finite or infinite) alphabet Γ of *types*. It will be reasonable to assume that $|\Gamma| \geqslant 2$. A *trace* (over Γ) is a string $t \in \Gamma^+$. We write $\mathit{types}(t)$ for the set of types that occur in t.

This paper studies a basic class of event queries which we call *subsequence-queries with wildcards and gap-size constraints*, for short: *swg-queries*. For defining these queries, we fix a countably infinite set Vars of *variables*, and we will always assume that Vars is disjoint with the set Γ of considered types. An *swg-query* q (over Vars and Γ) is specified

by a *query string* $s \in (\mathsf{Vars} \cup \Gamma)^+$, a *global window size* $w \in \mathbb{N}_{\geqslant 1} \cup \{\infty\}$ with $w \geqslant |s|$, and a tuple $c = (c_1, c_2, \ldots, c_{|s|-1})$ *of local gap-size constraints* (*for* $|s|$ *and* w), where $c_i = (c_i^-, c_i^+) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, such that $c_i^- \leqslant c_i^+$ for every $i \in [|s|{-}1]$ and $|s| + \sum_{i=1}^{|s|} c_i^- \leqslant w$. We denote such queries in the form $q = (s, w, c)$. Note that setting all gap-size constraints of a swg-query q to $(0, \infty)$ corresponds to a query without gap-size constraints.

We write *types*(q) (or *types*(s)) and *vars*(q) (or *vars*(s)) for the set of types and the set of variables, respectively, that occur in q's query string s. A query q is called an (ℓ, w, c)*-query* (over Vars and Γ) if $q = (s, w, c)$ with $|s| = \ell$; the parameter ℓ will be called *string length*. We write $\mathcal{Q}$ to denote the class of all swg-queries.

The semantics of swg-queries is defined as follows: Each variable in s serves as a *wildcard* representing an arbitrary type. A query $q = (s, w, c)$ *matches in* a trace t if the wildcards in s can be replaced by types in such a way that the resulting string s' satisfies the following: t contains a factor t' of length at most w such that s' occurs as a subsequence in t' and for each $i < \ell := |s|$ the gap between $s'[i]$ and $s'[i{+}1]$ in t' has length at least c_i^- and at most c_i^+. I.e., t' is of the form $s'[1]\, g_1\, s'[2]\, g_2 \cdots g_{\ell-1}\, s'[\ell]$ and $c_i^- \leqslant |g_i| \leqslant c_i^+$ for all $i \in [\ell{-}1]$.

An alternative description of these semantics, which will be more convenient for our formal proofs, involves a bit more notation: We say that an embedding $e : [\ell] \to [n]$ *satisfies* a global window size w, if $e(\ell) - e(1) + 1 \leqslant w$; and we say that e *satisfies* a tuple $c = (c_1, c_2, \ldots, c_{\ell-1})$ of local gap-size constraints (for ℓ and w), if $c_i^- \leqslant e(i{+}1){-}1 - e(i) \leqslant c_i^+$ for all $i < \ell$.

A *substitution* is a mapping $\mu : (\mathsf{Vars} \cup \Gamma) \to (\mathsf{Vars} \cup \Gamma)$ with $\mu(\gamma) = \gamma$ for all $\gamma \in \Gamma$. We extend substitutions to mappings $(\mathsf{Vars} \cup \Gamma)^+ \to (\mathsf{Vars} \cup \Gamma)^+$ in the obvious way, i. e., $\mu(s) = \mu(s[1])\mu(s[2]) \cdots \mu(s[\ell])$ for $s \in (\mathsf{Vars} \cup \Gamma)^+$ and $\ell := |s|$. We sometimes also consider partial substitutions of the form $(V \cup \Delta) \to (V' \cup \Delta)$ for some $V, V' \subseteq \mathsf{Vars}$ and $\Delta \subseteq \Gamma$.

An swg-query $q = (s, w, c)$ *matches in* a trace $t \in \Gamma^+$ (or, t *matches* q, in symbols: $t \models q$), if and only if there are a substitution $\mu : (\mathsf{Vars} \cup \Gamma) \to \Gamma$ and an embedding $e : [|s|] \to [|t|]$ that satisfies w and c, such that $\mu(s) \preccurlyeq_e t$. We call (μ, e) *a witness for* $t \models q$.

▶ **Example 2.1.** Let $x_1, x_2, x_3 \in \mathsf{Vars}$ and $\Gamma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. We consider a query $q = (s, w, c)$, where $s = x_1\,\mathsf{a}\,\mathsf{b}\,x_1\,\mathsf{b}\,x_2\,\mathsf{c}\,x_3\,\mathsf{a}\,x_1$, $w = 25$ and $c = ((0,1),(0,0),(2,\infty),(4,\infty),(0,5),(0,5),(0,5),(1,5),(2,3))$, and a trace $t = t_1\,\mathsf{c}\,\mathsf{a}\,\mathsf{b}\,\mathsf{b}\,\mathsf{b}\,\mathsf{c}\,\mathsf{a}\,\mathsf{b}\,\mathsf{a}\,\mathsf{c}\,\mathsf{a}\,\mathsf{b}\,\mathsf{a}\,\mathsf{c}\,\mathsf{b}\,\mathsf{c}\,\mathsf{a}\,\mathsf{b}\,\mathsf{a}\,\mathsf{c}\,t_2$, where $t_1, t_2 \in \Gamma^*$. We observe that $t \models q$, and a witness substitution and embedding can be illustrated as follows:

$$\begin{array}{lllllllllllllllllll} s & = & & x_1 & \mathsf{a}\,\mathsf{b} & & x_1 & & & \mathsf{b} & x_2 & \mathsf{c} & x_3 & & \mathsf{a} & & x_1\,, & & \\ t & = & t_1 & \mathsf{c} & \mathsf{a}\,\mathsf{b} & \mathsf{b}\,\mathsf{b} & \mathsf{c} & \mathsf{a}\,\mathsf{b}\,\mathsf{a}\,\mathsf{c}\,\mathsf{a} & & \mathsf{b} & \mathsf{a} & \mathsf{c} & \mathsf{b} & \mathsf{c} & \mathsf{a} & \mathsf{b}\,\mathsf{a} & \mathsf{c} & t_2\,. \end{array}$$

Recall that in the introduction we have presented another, more practical scenario that can also be described by an swg-query (see Example 1.1).

The *model set* of a query q w.r.t. a type set $\Delta \subseteq \Gamma$ is $\mathsf{Mod}_\Delta(q) := \{\, t \in \Delta^+ \,:\, t \models q \,\}$. For two queries q and q' and a set $\Delta \subseteq \Gamma$ we say that q *is contained in* q' w.r.t. Δ if $\mathsf{Mod}_\Delta(q) \subseteq \mathsf{Mod}_\Delta(q')$; and we say that q *and* q' *are equivalent w.r.t.* Δ if $\mathsf{Mod}_\Delta(q) = \mathsf{Mod}_\Delta(q')$. The remainder of this section is devoted to a characterisation of containment and equivalence of (ℓ, w, c)-queries via suitable notions of homomorphisms and isomorphisms. Let us fix an $\ell \in \mathbb{N}_{\geqslant 1}$, a $w \in \mathbb{N} \cup \{\infty\}$ with $w \geqslant \ell$, and a tuple c of local gap-size constraints for ℓ and w.

▶ **Definition 2.2.** A *homomorphism* from an (ℓ, w, c)-query $q' = (s', w, c)$ to an (ℓ, w, c)-query $q = (s, w, c)$ is a substitution $h : (\mathsf{Vars} \cup \Gamma) \to (\mathsf{Vars} \cup \Gamma)$ such that $h(s') = s$ (i.e., $h(s'[i]) = s[i]$ for all $i \in [\ell]$). We write $q' \xrightarrow{\text{hom}} q$ to express that there exists a homomorphism from q' to q.

We next characterise containment via homomorphisms (see item (c) of the next theorem).

▶ **Theorem 2.3.** *Let q and q' be (ℓ, w, c)-queries over* Vars *and* Γ.
(a) *If* $q' \xrightarrow{\text{hom}} q$ *then* $types(q') \subseteq types(q)$ *and* $\mathsf{Mod}_\Gamma(q') \supseteq \mathsf{Mod}_\Gamma(q)$.
(b) *Let* $\Delta \subseteq \Gamma$ *be such that* $|\Delta| \geqslant 2$ *and* $\Delta \supseteq types(q)$. *If* $\mathsf{Mod}_\Delta(q') \supseteq \mathsf{Mod}_\Delta(q)$ *then* $q' \xrightarrow{\text{hom}} q$.
(c) *If* $|\Gamma| \geqslant 2$, *then* $\mathsf{Mod}_\Gamma(q') \supseteq \mathsf{Mod}_\Gamma(q) \iff q' \xrightarrow{\text{hom}} q$.

Let us consider an example application of this characterisation of containment.

▶ **Example 2.4.** Let us fix ℓ distinct variables $x_1, \ldots, x_\ell \in \mathsf{Vars}$, and let $s_{mg} := x_1 x_2 \cdots x_\ell$. The query $q_{mg} = (s_{mg}, w, c)$ is the *most general* (ℓ, w, c)-*query* in the sense that $\mathsf{Mod}_\Gamma(q_{mg}) \supseteq \mathsf{Mod}_\Gamma(q)$ for every (ℓ, w, c)-query $q = (s, w, c)$. To see this, note that the mapping defined via $h(x_i) := s[i]$ for all $i \in [\ell]$ provides a homomorphism from q_{mg} to q and use Theorem 2.3(a).

One might ask why Theorem 2.3 restricts attention to queries q and q' of the same parameters (ℓ, w, c). One answer is that this is exactly what is needed for our purposes in Section 3: to ensure that our algorithm for query discovery always produces *meaningful* results, we precisely need this characterisation (see also Remark 3.2). Another answer is that, when considering queries q and q' with different parameters (ℓ, w, c) and (ℓ', w', c'), a characterisation analogous to the one provided by Theorem 2.3(c) simply is not available:

▶ **Example 2.5.** For the sake of this example, assume that Definition 2.2 applies to arbitrary queries $q = (s, w, c)$ and $q' = (s', w', c')$ of the same query string length $\ell := |s| = |s'|$ but where the global window sizes and the local gap-size constrains may differ from each other.

First consider two swg-queries $q = (s, w, c)$ and $q' = (s, w, c')$ having the same query string $s = x\,\mathsf{a}\,x$ and the same global window size $w = 4$, but different local gap-size constraints $c = ((0,1),(0,0))$ and $c' = ((0,0),(0,0))$. It holds that $q' \xrightarrow{\text{hom}} q$, but $\mathsf{Mod}_\Gamma(q') \not\supseteq \mathsf{Mod}_\Gamma(q)$, e. g., $\mathsf{bcab} \in \mathsf{Mod}_\Gamma(q) \setminus \mathsf{Mod}_\Gamma(q')$.

Almost the same example queries show that the characterisation does also not hold for queries with the same query string length and the same gap-size constraints but different global window sizes: Let $q = (s, w, c)$ and $q' = (s, w', c)$ with $s = x\,\mathsf{a}\,x$, $c = ((0,\infty),(0,\infty))$, and $w = 4$ and $w' = 3$. Again, we can observe that $q' \xrightarrow{\text{hom}} q$, but $\mathsf{Mod}_\Gamma(q') \not\supseteq \mathsf{Mod}_\Gamma(q)$ (witnessed by the same trace bcab). Note furthermore, that tuples of gap-size constraints of the form $((0,\infty),\ldots,(0,\infty))$ can even be considered as not having any gap-size constraints.

There are natural candidates of extending the concept of homomorphisms to the case of queries with different query string lengths. For example, we could adapt Definition 2.2 by requiring the substitution $h : (\mathsf{Vars} \cup \Gamma) \to (\mathsf{Vars} \cup \Gamma)$ to satisfy that $h(s')$ is a subsequence of s. However, it can be easily seen that for $q := (s, w, c)$ and $q' := (s', w', c')$ with $s = x\,\mathsf{a}\,\mathsf{b}\,x$, $s' = x\,\mathsf{a}\,x$, $w = 4$, $w' = 3$, $c = ((0,\infty),(0,\infty),(0,\infty))$, and $c' = ((0,\infty),(0,\infty))$, we have $q' \xrightarrow{\text{hom}} q$ (with respect to the adapted definition), but $\mathsf{c}\,\mathsf{a}\,\mathsf{b}\,\mathsf{c} \in \mathsf{Mod}_\Gamma(q) \setminus \mathsf{Mod}_\Gamma(q')$.

▶ **Definition 2.6.** Two (ℓ, w, c)-queries $q = (s, w, c)$ and $q' = (s', w, c)$ are called *isomorphic* (denoted by $q \cong q'$), if there is a bijection $\pi : (vars(q) \cup \Gamma) \to (vars(q') \cup \Gamma)$ such that $\pi(\gamma) = \gamma$ for all $\gamma \in \Gamma$ and $\pi(s[i]) = s'[i]$ for all $i \in [\ell]$.

▶ **Corollary 2.7.** *Let* $|\Gamma| \geqslant 2$. *For all* (ℓ, w, c)-*queries* q *and* q' *over* Γ *and* Vars *we have:*
$q \cong q' \iff \big(q \xrightarrow{\text{hom}} q' \text{ and } q' \xrightarrow{\text{hom}} q \big) \iff \mathsf{Mod}_\Gamma(q) = \mathsf{Mod}_\Gamma(q')$.
Furthermore, upon input of q *and* q' *we can decide in time* $\mathrm{O}(\ell)$ *whether or not* $q \cong q'$.

The following notion of "partially isomorphic" queries will be useful for Section 3.

▶ **Definition 2.8.** Let $q = (s, w, c)$ and $q' = (s', w, c)$ be (ℓ, w, c)-queries and let $I \subseteq [\ell]$. We say that q *is partially isomorphic to* q' *w.r.t.* I (and shortly write $q \sim_I q'$) if, and only if,
1. for all $i, j \in I$ we have $\big(s[i] = s[j] \iff s'[i] = s'[j] \big)$, and
2. for all $i \in I$ we have $\big(s[i] \in \Gamma \iff s'[i] \in \Gamma \big)$ and $\big($ if $s[i] \in \Gamma$ then $s[i] = s'[i]$ $\big)$.

▶ **Lemma 2.9.** *For all* (ℓ, w, c)-*queries* q *and* q' *we have:* $q \sim_{[\ell]} q' \iff q \cong q'$.

3 Discovery of Descriptive (ℓ, w, c)-Queries

A *sample* is a finite, non-empty set $\mathcal{S}$ of traces over Γ. The *support* $\mathsf{supp}(q, \mathcal{S})$ of a query q in $\mathcal{S}$ is defined as the fraction of traces in the sample that match q, i.e., $\mathsf{supp}(q, \mathcal{S}) := \frac{|\{t \in \mathcal{S} \,:\, t \models q\}|}{|\mathcal{S}|}$. A *support threshold* is a rational number sp with $0 < \mathsf{sp} \leqslant 1$. A query q is said to *cover* a sample $\mathcal{S}$ *with support* sp if $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$.

▶ **Definition 3.1.** Let Q be a class of queries, let $\mathcal{S}$ be a sample, and let sp be a support threshold. Let $\ell \in \mathbb{N}_{\geqslant 1}$, let $w \in \mathbb{N} \cup \{\infty\}$ with $w \geqslant \ell$, and let c be a tuple of local gap-size constraints for ℓ and w. A query q is called *descriptive for* $\mathcal{S}$ *w.r.t.* $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ if q is an (ℓ, w, c)-query in Q, $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$, and there is no (ℓ, w, c)-query $q' \in \mathsf{Q}$ with $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ and $\mathsf{Mod}_\Gamma(q') \subsetneq \mathsf{Mod}_\Gamma(q)$.

Let us explain this notion on an intuitive level by considering the case $\mathsf{sp} = 1$. In this case, a query q that is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ satisfies $\mathcal{S} \subseteq \mathsf{Mod}_\Gamma(q)$, and there is no other (ℓ, w, c)-query $q' \in \mathsf{Q}$ that can be "wrapped around" $\mathcal{S}$ more tightly, i. e., in the sense that $\mathcal{S} \subseteq \mathsf{Mod}_\Gamma(q') \subsetneq \mathsf{Mod}_\Gamma(q)$. Therefore, among all (ℓ, w, c)-queries from Q, the query q can be considered as one of the best descriptors for $\mathcal{S}$.

▶ Remark 3.2. Throughout this section we assume that $|\Gamma| \geqslant 2$. Hence, by Theorem 2.3 and Corollary 2.7 we know that, for any query class $\mathsf{Q} \subseteq \mathcal{Q}$, an swg-query q is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ if, and only if, q is an (ℓ, w, c)-query in Q, $\mathsf{supp}(q, \mathcal{S}) \geqslant \mathsf{sp}$, and there is no (ℓ, w, c)-query $q' \in \mathsf{Q}$ with $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ and $q \xrightarrow{\text{hom}} q'$ and $q \not\cong q'$.

The following notions will be convenient throughout the rest of this section. For an (ℓ, w, c)-query $q = (s, w, c)$ and a symbol $z \in \mathsf{Vars} \cup \Gamma$ we let $pos(q, z)$ (and $pos(s, z)$) be the set of all positions i of s that carry the symbol z. I.e., $pos(q, z) = pos(s, z) = \{i \in [\ell] \,:\, s[i] = z\}$. Given a query string $s \in (\mathsf{Vars} \cup \Gamma)^\ell$, a set of positions $P \subseteq [\ell]$, and a symbol $z \in \mathsf{Vars} \cup \Gamma$, we write $s\langle P \mapsto z\rangle$ to denote the query string s' obtained from s by placing z on all positions $i \in P$. I.e., $s'[i] = z$ for all $i \in P$ and $s'[j] = s[j]$ for all $j \in [\ell] \setminus P$. Accordingly, for a query $q = (s, w, c)$ we let $q\langle P \mapsto z\rangle$ be the query $(s\langle P \mapsto z\rangle, w, c)$. For a variable $x \in vars(q)$ we shortly write $q\langle x \mapsto z\rangle$ (and $s\langle x \mapsto z\rangle$) instead of $q\langle pos(q, x) \mapsto z\rangle$ (and $s\langle pos(s, x) \mapsto z\rangle$) – i.e., all occurrences of variable x in s are replaced by the symbol z.

Next, we define the problem of computing a descriptive query, and the problem to check for a given query whether it is descriptive; we will refer to these two problems as the *query discovery problems*. As our definition of descriptiveness (see Definition 3.1) depends on a class Q of swg-queries, the problems of computing a descriptive query or of checking whether a given query is descriptive are also parameterised by an arbitrary class Q of swg-queries. It will be discussed in more detail in Section 4 why this parameterisation by Q is vital.

Q-Compute Descriptive Query Problem (Q-CompDescQuery): The input is a sample $\mathcal{S}$ over Γ, a support threshold sp, a string length $\ell \in \mathbb{N}$, a global window size $w \geqslant \ell$, and a tuple c of local gap-size constraints (for ℓ and w). The task is to compute an (ℓ, w, c)-query $q \in \mathsf{Q}$ that is descriptive for $\mathcal{S}$ with respect to $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$.

Q-Check Descriptiveness Problem (Q-CheckDescQuery): The input is the same as for Q-CompDescQuery, but in addition also an (ℓ, w, c)-query $q \in \mathsf{Q}$, and the task is to decide whether q is descriptive for $\mathcal{S}$ with respect to $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$.

▶ Remark 3.3. Since a Q-CompDescQuery- or Q-CheckDescQuery-instance contains at most $2(|s|-1)+1$ integers (i. e., w and c_i^-, c_i^+ for every $i \in [|s|-1]$), running times polynomial in w, c_i^+ or c_i^- can be exponential in the input size. We deal with this issue by assuming, without

loss of generality, that $w = \infty$ or $w \leqslant \max\{|t| \colon t \in \mathcal{S}\}$, $c_i^+ = \infty$ or $c_i^+ \leqslant \max\{|t| \colon t \in \mathcal{S}\}$ for every $i \in [|s|-1]$, and $\sum_{i=1}^{\ell} c_i^- \leqslant \min\{w, \max\{|t| \colon t \in \mathcal{S}\}\}$. I.e., all integers of the input are bounded by $\max\{|t| \colon t \in \mathcal{S}\}$ and therefore their values are polynomial in the total input size.

3.1 An Algorithm for Solving the Query Discovery Problems

We now present an algorithm that, for suitable classes Q of queries, can be used to solve the problems Q-CompDescQuery and Q-CheckDescQuery. As input, the algorithm receives $\mathcal{S}$, sp, (ℓ, w, c), and an "initial" query $q_0 = (s_0, w, c) \in \mathsf{Q}$. The goal is to compute an (ℓ, w, c)-query $q \in \mathsf{Q}$ that is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ and that satisfies $\mathsf{Mod}_\Gamma(q) \subseteq \mathsf{Mod}_\Gamma(q_0)$.

If $\mathsf{supp}(q_0, \mathcal{S}) < \mathsf{sp}$, then such a query q does not exist because of the following reasoning: obviously, if q_0 does not describe $\mathcal{S}$ with sufficient support, then no other query q' with $\mathsf{Mod}_\Gamma(q') \subseteq \mathsf{Mod}_\Gamma(q_0)$ can describe $\mathcal{S}$ with sufficient support.

If $\mathsf{supp}(q_0, \mathcal{S}) \geqslant \mathsf{sp}$, then the algorithm proceeds as follows. It considers the variables $\mathit{vars}(q_0) = \{x_1, x_2, \ldots, x_k\}$ in some arbitrary order. For each x_i, it performs the following *replacement operation*: choose a symbol y (which can be a type or a variable) and replace *all* occurrences of x_i in the current query string by y. However, such a replacement operation is only admissible if the new query q' is in Q and satisfies $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$. If no admissible replacement operation is possible, then x_i is not replaced, i. e., the current query string is not changed. After each variable $x_i \in \mathit{vars}(q_0)$ has been considered, the algorithm terminates and produces the current query as output.

This means that the algorithm produces a sequence $s_0, s_1, \ldots, s_k$ of query strings, where, for every $i \in [k]$, either s_i can be obtained from s_{i-1} by a replacement operation, or $s_i = s_{i-1}$. For every $i \in [k]$, let $q_i = (s_i, w, c)$ be the (ℓ, w, c)-query that corresponds to s_i. We note that $q_0 \xrightarrow{\text{hom}} q_1 \xrightarrow{\text{hom}} \ldots \xrightarrow{\text{hom}} q_k$ and therefore, as a consequence of Theorem 2.3, $\mathsf{Mod}_\Gamma(q_0) \supseteq \mathsf{Mod}_\Gamma(q_1) \supseteq \ldots \supseteq \mathsf{Mod}_\Gamma(q_k)$. Hence, the algorithm starts with q_0 and then follows a path of length at most k in the search space of (ℓ, w, c)-queries from Q that cover $\mathcal{S}$ with support sp, always going from one query to a more specific one. The crucial point is now that if the replacement operations are done in a certain way, then it can be guaranteed that the final query q_k is necessarily *descriptive* for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. In other words, from any (ℓ, w, c)-query $q \in \mathsf{Q}$ that covers our sample with sufficient support, we will reach with at most $|\mathit{vars}(q_0)|$ replacement operations a descriptive query.

Let us call a replacement operation a *type operation* or a *variable operation* if it replaces x_i by a type or by a variable, respectively. For simplicity, assume that the algorithm considers (and possibly replaces) the variables in the order $x_1, x_2, \ldots, x_k$. Assume that we are at the i^{th} step of the algorithm, i. e., variable x_i is considered and the goal is to obtain s_i from s_{i-1}. The symbol y we choose to replace for x_i can either be a type or a variable. For a type operation we can choose from all types that occur in sufficiently many traces of the sample $\mathcal{S}$. For a variable operation we can choose one of the variables x_j that have been considered before (i. e., $j < i$) and that have *not* been replaced (i. e., at the j^{th} step when we considered x_j, no replacement operation was admissible). Let us clarify this with an example.

Let $s_0 = x_2 x_1 \,\mathsf{a}\, x_1 \,\mathsf{b}\, x_3 x_4$, and let us assume that we first consider x_1. With the restriction mentioned above, we can initially only use type operations. For the sake of the example, assume further that no type operation for x_1 is admissible (since it would yield $q_1 \notin \mathsf{Q}$ or $\mathsf{supp}(q_1, \mathcal{S}) < \mathsf{sp}$); thus, $s_1 = s_0$. Next we consider variable x_2 and we note that since x_1 has already been considered and has not been replaced in the first step, $y := x_1$ is a valid candidate for a variable operation. Let us assume that it turns out that it also is admissible. Then, we can replace x_2 by x_1 and obtain $s_2 = x_1 x_1 \,\mathsf{a}\, x_1 \,\mathsf{b}\, x_3 x_4$. However, this also means that x_2 will never be available as a valid candidate for variable operations later on (which is also the case if a type operation had been used for replacing x_2 with an admissible type).

To see that this restriction is indeed necessary, assume the following example: The sample $\mathcal{S}$ consists of only one trace, namely the trace aaa, and we have ℓ=3, w=3, $c = ((0,0),(0,0))$, and $s_0 = x_1x_2x_3$. Without the restriction from above, we could obtain $s_3 = x_1x_1x_1$ with variable operations by replacing x_1, x_2 and x_3 by the same variable x_1. Note that replacing these variables by x_1 indeed violates our restriction, because when considering x_1 in the first step, replacing x_1 by the type a is in fact an admissible replacement operation and therefore we *have* to replace x_1 in the algorithm's first step (and hence in later steps the variable x_1 is not available). If we perform the replacement operations according to our restrictions, we in fact perform the type operation with $y := \mathsf{a}$ for each of the variables x_1, x_2, x_3, resulting in the query $q' = (\mathsf{aaa}, w, c)$. Note that the query $q = (x_1x_1x_1, w, c)$, which could be otained when ignoring our restriction, is *not* descriptive (since $\mathsf{Mod}_\Gamma(q') \subsetneq \mathsf{Mod}_\Gamma(q)$). This illustrates that we indeed need the restriction described above. Note that q' is actually the only descriptive query for the particular example considered here.

With the restricted kind of variable operations, we can show that the algorithm always produces descriptive queries, provided that the query class Q has suitable closure properties. Pseudocode of the algorithm is provided in Algorithm 1. Let us briefly explain how the pseudocode actually implements the idea outlined above. Initially, we collect in Δ the types available for type operations (note that if $\frac{|\{t \in \mathcal{S} \,:\, \gamma \in types(t)\}|}{|\mathcal{S}|} < \mathsf{sp}$, then the type γ cannot occur in any query that covers $\mathcal{S}$ with support sp), we initialise the set U of *unvisited* variables as the set of *all* variables of s_0, and we define $V := \emptyset$ as the (empty) set of variables that are initially available for variable operations. Now the main loop in line 5 considers each variable $x \in U$ exactly once, and for each such variable, checks whether there exists a type or variable y from $\Omega := (\Delta \cup V)$ such that replacing all occurrences of variable x by the symbol y is an admissible replacement operation. If such an $y \in \Omega$ exists, we perform the actual replacement in line 12. If no such element in Ω exists, then no replacement operation is possible, and we do not change the current query string but insert x into the set V of variables available for future variable operations (line 16).

Theorem 3.4 summarises the guarantees we can provide for Algorithm 1. For the precise statement we need two notions of closure properties that the query class Q has to satisfy: Q is called *closed under isomorphisms* if for all parameters (ℓ, w, c) and for every (ℓ, w, c)-query $q \in \mathsf{Q}$, the set Q also contains all (ℓ, w, c)-queries q' with $q \cong q'$. Q is called *closed under m-grained generalisations*, for an $m \in \mathbb{N}_{\geqslant 1} \cup \{\infty\}$, if the following is true for every query $q = (s, w, c) \in \mathsf{Q}$ and every set of positions $P \subseteq [|s|]$ with $|P| \leqslant m$ and $s[i] = s[j]$ for all $i, j \in P$: There is a variable $x \in \mathsf{Vars} \setminus vars(q)$ such that the query $q\langle P \mapsto x\rangle$ belongs to Q.

We explain this concept on an intuitive level by considering $m = 1$. A 1-grained generalisation takes a $q \in \mathsf{Q}$ and makes it more general by replacing the symbol at some position of the query string by a single occurrence of a new variable. E. g., $x\,\mathsf{a}\,yx\,\mathsf{b} \rightsquigarrow xz_1yx\,\mathsf{b} \rightsquigarrow xz_1z_2x\,\mathsf{b} \rightsquigarrow z_3z_1z_2x\,\mathsf{b}$ is a sequence of 1-grained generalisations, where z_1, z_2, z_3 are new variables. If for every $q \in \mathsf{Q}$ every 1-grained generalisation necessarily produces a query in Q, then Q is closed under 1-grained generalisations. For example, this is the case for the class $\mathcal{Q}^{|rv(s)|\leqslant k}$ of queries with at most k repeated variables, i.e. variables with at least two occurrences in s, or the class $\mathcal{Q}^{w\leqslant k}$ of queries with window size at most k, for every constant $k \in \mathbb{N}$. On the other hand, the class $\mathcal{Q}^{|vars(s)|\leqslant k}$ of queries with at most k variables is not closed under 1-grained generalisations. An m-grained generalisation replaces in one step at most m occurrences of *the same* symbol by occurrences of the same new variable. Since m-grained generalisations for $m \geqslant 2$ introduce new repeated variables, the class $\mathcal{Q}^{|rv(s)|\leqslant k}$ is not closed under m-grained generalisations. On the other hand, the class $\mathcal{Q}^{|s|\leqslant k}$, which consists only of queries with a query string length of at most k, is closed under

Algorithm 1 Q-DescrSWGQuery($\mathcal{S}$,sp, q_0).

Input : sample $\mathcal{S}$; support threshold sp with $0 < \mathsf{sp} \leqslant 1$; query $q_0 = (s_0, w, c)$ in Q
Returns : descriptive query q for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (|s_0|, w, c))$ or error message $\perp$

1 **if** $\mathsf{supp}(q_0, \mathcal{S}) < \mathsf{sp}$ **then stop and return** $\perp$
2 $\Delta := \{\gamma \in \Gamma \ : \ \frac{|\{t \in \mathcal{S} \,:\, \gamma \in types(t)\}|}{|\mathcal{S}|} \geqslant \mathsf{sp}\}$ // types to be considered
3 $s := s_0$; $q := (s, w, c)$ // query string and query
4 $U := vars(q_0)$; $V := \emptyset$ // unvisited variables and available variables
5 **while** $U \neq \emptyset$ **do**
6 **select** an arbitrary $x \in U$ and let $U := U \setminus \{x\}$
7 $\Omega := (\Delta \cup V)$; replace := False
8 **while** $\Omega \neq \emptyset$ **do**
9 **select** an arbitrary $y \in \Omega$ and let $\Omega := \Omega \setminus \{y\}$
10 $q' := (s\langle x \mapsto y\rangle, w, c)$
11 **if** $q' \in \mathsf{Q}$ **and** $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ **then**
12 $s := s\langle x \mapsto y\rangle$ // ReplaceOp
13 replace := True
14 **break** inner loop
15 **if** replace *is* False **then**
16 $V := V \cup \{x\}$ // NoChangeOp
17 **stop and return** $q := (s, w, c)$

∞-grained generalisations. Since $q' \xrightarrow{\text{hom}} q$ for every query q' obtained from q by an m-grained generalisation, Theorem 2.3(a) implies that q' is actually a more general query in the sense that $\mathsf{Mod}_\Gamma(q') \supseteq \mathsf{Mod}_\Gamma(q)$.

We are now ready for the formal statement of this subsection's main theorem. Note that the lines 6 and 9 of Algorithm 1 allow to make an arbitrary choice. Different choices lead to different "runs" of the algorithm, and different runs might produce different output queries (depending on the particular order in which the elements in $vars(q_0)$ and in Ω are considered). The next theorem states that *any* choice is fine, and that *any* output query is guaranteed to be a descriptive query.

▶ **Theorem 3.4.** *Let $|\Gamma| \geqslant 2$ and let Q be a class of swg-queries over Γ and* Vars *that is closed under isomorphisms. Let $\mathcal{S}$ be a sample, let* sp *be a support threshold, and let $q_0 = (s_0, w, c)$ be a query in Q. Let $\ell = |s_0|$ and let $m \in \mathbb{N}$ be such that $|pos(q_0, x)| \leqslant m$ for all $x \in vars(q_0)$.*

(a) *If $\mathsf{supp}(q_0, \mathcal{S}) < \mathsf{sp}$, then there does not exist any descriptive query q' for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ such that $\mathsf{Mod}_\Gamma(q') \subseteq \mathsf{Mod}_\Gamma(q_0)$, and there is only one run of algorithm Q-DescrSWGQuery($\mathcal{S}$,sp, q_0), and this run stops in line 1 with output $\perp$.*

(b) *If $\mathsf{supp}(q_0, \mathcal{S}) \geqslant \mathsf{sp}$ and Q is closed under m-grained generalisations, then every run of Q-DescrSWGQuery($\mathcal{S}$,sp, q_0) terminates and outputs a query q' that is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$ and satisfies $\mathsf{Mod}_\Gamma(q') \subseteq \mathsf{Mod}_\Gamma(q_0)$; furthermore, $q' = q_0$ if, and only if, q_0 is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$.*

The proof of (a) is obvious while the proof of (b) is quite demanding and proceeds as follows. For simplicity, let us focus here on the case that $\mathsf{Q} = \mathcal{Q}$. We have already observed above that the output query $q' := q_k$ is necessarily an (ℓ, w, c)-query from Q with $\mathsf{supp}(q_k, \mathcal{S}) \geqslant \mathsf{sp}$. Now, for contradiction, let us assume that q_k is not descriptive for $\mathcal{S}$ with respect to $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. Then, according to Definition 3.1, there exists an (ℓ, w, c)-query $\tilde{q} \in \mathsf{Q}$ with support $\mathsf{supp}(\tilde{q}, \mathcal{S}) \geqslant \mathsf{sp}$ and $\mathsf{Mod}_\Gamma(\tilde{q}) \subsetneq \mathsf{Mod}_\Gamma(q_k)$. This means that $q_k \xrightarrow{\text{hom}} \tilde{q}$ and $\tilde{q} \not\cong q_k$ (see

Remark 3.2). We can now inductively show that for every $i \in \{0, \dots, k\}$ the queries q_i and $\tilde{q}$ are partially isomorphic (see Definition 2.8) with respect to the positions of the variables already considered by the algorithm. For $i = k$ this yields $\tilde{q} \cong q_k$; a contradiction. Proving the induction step is non-trivial and constitutes the most crucial technical contribution of the correctness proof. In particular, we heavily rely on the technical machinery developed in Section 2. A detailed proof will be included in the full version of this paper.

We now discuss how the algorithm can be used to solve the problems Q-CompDescQuery and Q-CheckDescQuery. For solving the problem Q-CompDescQuery, we only need that Q is closed under isomorphisms and 1-grained generalisations, since then, we can run it with $s_0 = x_1 x_2 \dots x_\ell$ (i. e., the most general (ℓ, w, c)-query, see Example 2.4) in order to compute a query that is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. On the other hand, if we want to solve Q-CheckDescQuery for some (ℓ, w, c)-query $q \in \mathsf{Q}$, then we can run the algorithm with $q_0 := q$, and then check whether its output equals q_0. The latter is the case if, and only if, q is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. However, this only works for queries $q = (s, w, c)$ such that Q is closed under m-grained generalisations, where $m \in \mathbb{N}$ is such that $|pos(q, x)| \leqslant m$ for all $x \in vars(q)$.

Let us close this subsection by an outlook on future work. Clearly, a single run of Algorithm 1 only outputs a single query that is descriptive w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. However, by exploring all possible runs it is possible to enumerate a large number of queries that are descriptive w.r.t. $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. But care must be taken when exploring the search space of all possible runs, since two different runs of Algorithm 1 may lead to the same output query and we want to avoid outputting the same query twice. We plan to address this enumeration task and suitable selection strategies in our future work. An obvious question is if *every* descriptive query can be produced by a suitable run of Algorithm 1. The answer is "no", as the following example shows. Given the sample $\mathcal{S} = \{\mathsf{aab}, \mathsf{abb}\}$, $\mathsf{sp} = 1$ and $q_0 = (x_1 x_2, w, c)$ with $w = 2$ and $c = ((0, \infty))$, *every* run of Algorithm 1 outputs the descriptive query $q := (\mathsf{ab}, w, c)$. But note that also the query $q' := (x_i x_i, w, c)$ (for an arbitrary $i \in \{1, 2\}$) is descriptive for $\mathcal{S}$ w.r.t. $(\mathsf{Q}, \mathsf{sp}, (2, w, c))$. However, there does not exist any run of Algorithm 1 that outputs q'. Obvious future tasks are (a) to find a precise characterisation of the descriptive queries that can be produced by a run of Algorithm 1, and (b) to design an efficient algorithm that is capable of producing *all* descriptive queries.

3.2 The Complexity of the Query Discovery Problems

In Algorithm 1, we treat the checks of $q' \in \mathsf{Q}$ and $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ in line 11 as black-box requests to an oracle. Clearly, we can check if $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ by solving the following *matching problem* for $|\mathcal{S}|$ times. For any fixed query class $\mathsf{Q} \subseteq \mathcal{Q}$, the **Q-Matching Problem (Q-Match)** receives as input a query $q = (s, w, c) \in \mathsf{Q}$ and a trace t. The task is to decide if $t \models q$.

Since Q-Match-instances contain integers, we make, w.l.o.g., the same assumptions as described in Remark 3.3: we assume $w = \infty$ or $w \leqslant |t|$, $c_i^+ = \infty$ or $c_i^+ \leqslant |t|$ for every $i \in [|s|-1]$, and $\sum_{i=1}^{\ell} c_i^- \leqslant \min\{w, |t|\}$. For the following complexity analysis, let us call the oracles that check $q' \in \mathsf{Q}$ and solve Q-Match the Q-*membership oracle* and Q-*match oracle*, respectively. A straightforward analysis of Algorithm 1 yields the following.

▶ **Theorem 3.5.** *Every run of* Q-DescrSWGQuery($\mathcal{S}$,sp, q_0) *performs* $\mathrm{O}(|vars(q_0)|(|types(\mathcal{S})| + |vars(q_0)|)(\ell + |\mathcal{S}|))$ *computational steps,* $\mathrm{O}(|vars(q_0)|(|types(\mathcal{S})| + |vars(q_0)|))$ *calls to the* Q-*membership oracle and* $\mathrm{O}(|vars(q_0)|(|types(\mathcal{S})| + |vars(q_0)|)|\mathcal{S}|)$ *calls to the* Q-*match oracle.*

Theorem 3.5 shows that apart from answering the oracle requests, the complexity of algorithm Q-DescrSWGQuery is rather low. For most natural classes of queries Q, it is a reasonable assumption that checking $q \in \mathsf{Q}$ for abitrary queries $q \in \mathcal{Q}$ is a computationally simple task (i. e., solvable in time linear or quadratic in $|s|$). Hence, not much complexity seems to be hidden in the Q-membership oracle. On the other hand, as shall be discussed in Section 4, Q-Match is NP-complete for many classes Q; thus, substantial computational complexity is hidden in the Q-match oracle.

Upper bounds for Q-Match directly yield upper bounds for the query discovery problems via Theorem 3.5. On the other hand, the intractability of Q-Match is only an obstacle for efficient query discovery w.r.t. algorithms that are based on solving the matching problem (like Q-DescrSWGQuery). However, as we shall demonstrate next, the matching problem can be reduced in polynomial time to our query discovery problems; thus, *every* algorithm for one of the latter problems necessarily implicitly also solves the matching problem. This implies that lower bounds on the complexity of the matching problem carry over to lower bounds on the complexity of our query discovery problems. Consequently, the problem Q-Match (and its complexity) is central for query discovery.

The reduction from Q-Match to our query discovery problems is based on the following construction. Let $\mathsf{Q} \subseteq \mathcal{Q}$ be an arbitrary set of queries that is closed under isomorphisms and assume that $|\Gamma| \geqslant 2$. Let $q = (s, w, c) \in \mathsf{Q}$ be an (ℓ, w, c)-query with $\mathit{vars}(q) = \{x_1, x_2, \ldots, x_k\}$, and let t be a trace. Let $\delta, \delta' \in \Gamma$ with $\delta \neq \delta'$. For every $i \in [k]$, let μ_{x_i} be the substitution that maps x_i to δ and all other variables to δ'. Let $\mu_{\delta'}$ be the substitution defined by $\mu_{\delta'}(x_j) = \delta'$ for every $j \in [k]$. For every $z \in \{x_1, x_2, \ldots, x_k, \delta'\}$, let t_z be a trace over Γ defined by

$$t_z \;:=\; \mu_z(s[1])\; g_1\; \mu_z(s[2])\; g_2\; \cdots\; \mu_z(s[\ell{-}1])\; g_{\ell-1}\; \mu_z(s[\ell])\,,$$

where, for every $i < \ell$, g_i is the "gap string" consisting of c_i^- copies of some type from Γ. We now define the sample $\mathcal{S}_{q,t} := \{t, t_{\delta'}, t_{x_1}, t_{x_2}, \ldots, t_{x_k}\}$. By construction, for every $t' \in \mathcal{S} \setminus \{t\}$, $t' \models q$ is witnessed by the same embedding. We can show the following.

▶ **Theorem 3.6.** *The following statements are equivalent:*

1. $t \models q$,
2. $q \cong q'$ *for every* (ℓ, w, c)*-query* q' *that is descriptive for* $\mathcal{S}_{q,t}$ *with respect to* $(\mathsf{Q}, 1, (\ell, w, c))$,
3. q *is descriptive for* $\mathcal{S}_{q,t}$ *with respect to* $(\mathsf{Q}, 1, (\ell, w, c))$.

Theorem 3.6 reduces Q-Match to Q-CompDescQuery and Q-CheckDescQuery as follows. We can check $t \models q$ for a given (ℓ, w, c)-query q and a trace t by computing an (ℓ, w, c)-query q' that is descriptive for $\mathcal{S}_{q,t}$ with respect to $(\mathsf{Q}, 1, (\ell, w, c))$ and then checking $q \cong q'$, or by checking whether q is descriptive for $\mathcal{S}_{q,t}$ with respect to $(\mathsf{Q}, 1, (\ell, w, c))$. The next section is devoted to a detailed study of the complexity of Q-Match.

4 The Complexity of the Matching Problem

The algorithm presented in Section 3 computes a query that is descriptive for a sample $\mathcal{S}$ with respect to $(\mathsf{Q}, \mathsf{sp}, (\ell, w, c))$. This algorithm is based on the matching problem, i. e., it needs a sub-routine that solves Q-Match: check whether $t \models q$ for a given query $q \in \mathsf{Q}$ and a given trace t. As discussed at the end of Section 3, the algorithm only performs a polynomial number of such calls to a Q-match oracle, so the only source of exponential complexity might be hidden in the sub-routine solving Q-Match. Moreover, as demonstrated by Theorem 3.6 and the explanations following this result, the potential intractability of Q-Match inevitably

makes not only the algorithm presented in Section 3 inefficient, but *every* algorithm that computes descriptive queries. In this section we show that the matching problem for general swg-queries is NP-complete. In order to identify tractable subclasses, it therefore is absolutely vital that our algorithm from Section 3 can be parameterised by an arbitrary class Q of queries, since then it does not need to solve the matching problem for arbitrary queries, but only for queries from Q, for which the matching problem might be tractable.

In other words: From Section 3.2 we know that computing descriptive queries with respect to Q is tractable if, and only if, the matching problem for the class Q is tractable. Therefore, the complexity of the problem Q-Match is crucial for the complexity of computing descriptive queries. This section is devoted to a comprehensive classical and parameterised complexity analysis of the matching problem.

Recall from Section 2 that $\mathcal{Q}$ denotes the class of all swg-queries. The input of the matching problem consists of a swg-query $q = (s, w, c)$ and a trace t. We consider the parameters shown in Figure 2, where $\mathit{repvars}(q) = \mathit{repvars}(s)$ is the set of repeated variables (i. e., variables with at least two occurrences in s), $\mathit{maxgaps}(c) = \max\{c_i^+ - c_i^- + 1 \colon i \in [|s|-1]\}$ and $\mathit{maxc}^+(c) = \max\{c_i^+ \colon i \in [|s|-1]\}$. Whenever we consider w as a parameter, we will assume that $w \neq \infty$; and if we consider $\mathit{maxgaps}(c)$ or $\mathit{maxc}^+(c)$ as a parameter, we will assume that $c_i^+ \neq \infty$ for every $i \in [|s|-1]$.

$\lvert t\rvert$	length of the trace t,	$\mathit{maxc}^+(c)$	maximum upper local gap-size,
w	global window size,	$\lvert\mathit{repvars}(s)\rvert$	number of repeated variables in s,
$\lvert s\rvert$	length of query string,	$\lvert\mathit{types}(t)\rvert$	number of types in the trace t,
		$\mathit{maxgaps}(c)$	maximum gap size.

Figure 2 Parameters of the problem $\mathcal{Q}$-Match.

For studying restricted or parameterised variants of $\mathcal{Q}$-Match, we use the following terminology. Let $p_1, p_2, \ldots$ be some of the parameters defined above. By $\mathcal{Q}$-Match *parameterised by* $(p_1, p_2, \ldots)$, we denote the parameterised variant of $\mathcal{Q}$-Match that arises from considering $p_1, p_2, \ldots$ as parameters. Moreover, if we write $p_i \leqslant k$ for some parameter p_i and some constant $k \in \mathbb{N}$, then we consider the problem variant were parameter p_i is bounded by the constant k. If all mentioned parameters are bounded by a constant, then the resulting problem variant is not a parameterised problem and therefore we write $\mathcal{Q}$-Match *with* $(p_1 \leqslant k_1, p_2 \leqslant k_2, \ldots)$. For example, by $\mathcal{Q}$-Match *parameterised by* $(|s|, |\mathit{types}(t)|, \mathit{maxgaps}(c) \leqslant 3)$ we denote the parameterised variant of $\mathcal{Q}$-Match where $|s|$ and $|\mathit{types}(t)|$ are parameters, and inputs are restricted to instances satisfying $\mathit{maxgaps}(c) \leqslant 3$; and by $\mathcal{Q}$-Match *with* $(|\mathit{types}(t)| \leqslant 10, \mathit{maxgaps}(c) \leqslant 3)$ we denote the classical decision problem $\mathcal{Q}$-Match where the input is restricted to instances satisfying $|\mathit{types}(t)| \leqslant 10$ and $\mathit{maxgaps}(c) \leqslant 3$.

We start by studying the complexity of the classical (non-parameterised) problems: For all subsets $p_1, p_2, \ldots, p_r$ of the parameters defined above and for all constants $k_1, k_2, \ldots, k_r \in \mathbb{N}$, we aim at answering the following question:

▶ **Question 4.1.** *Is* $\mathcal{Q}$-Match *with* $(p_1 \leqslant k_1, p_2 \leqslant k_2, \ldots, p_r \leqslant k_r)$ *in* P*, or is it* NP*-hard?*

With respect to parameters $|t|$, w, $|s|$, and $|\mathit{repvars}(s)|$, answers to Question 4.1 can be obtained with moderate effort:

▶ **Theorem 4.2.** *For every constant* $k \in \mathbb{N}$*, each of the following problems is in* P*:* $\mathcal{Q}$-Match *with* $|t| \leqslant k$*,* $\mathcal{Q}$-Match *with* $w \leqslant k$*,* $\mathcal{Q}$-Match *with* $|s| \leqslant k$*,* $\mathcal{Q}$-Match *with* $|\mathit{repvars}(s)| \leqslant k$*.*

Proof sketch. A natural brute-force approach is as follows: Upon input of an swg-query $q = (s, w, c)$ and a trace t, we enumerate all mappings $\pi : \mathit{repvars}(q) \to \mathit{types}(t)$, and for each such mapping, we construct a regular expression R_π that describes all traces t' for which there exists a substitution $\mu : \mathit{vars}(q) \cup \Gamma \to \Gamma$ such that μ is an extension of π and $\mu(s) \preccurlyeq_e t'$ for some embedding e that satisfies w and c. Then, we only have to check for each of these mappings π, if the regular expression R_π matches in t. Another approach is to enumerate all embeddings $e : [|s|] \to [|t|]$ that satisfy w and c, and check for each such embedding e whether $\mu(s) \preccurlyeq_e t$ for some substitution μ (which can be done in time $\mathrm{O}(|s|)$, since μ must satisfy $\mu(s) = t[e(1)]t[e(2)] \dots t[e(|s|)]$). From these two algorithms and the obvious dependencies between the parameters, we can directly conclude the statements of the theorem. ◀

Next, we prove the following theorem. Observe that Theorems 4.3 and 4.2 answer Question 4.1 with respect to every subset $p_1, p_2, \dots, p_r$ of the parameters.

▶ **Theorem 4.3.** *Let $k_1, k_2, k_3 \in \mathbb{N}$. $\mathcal{Q}$-Match with $|\mathit{types}(t)| \leqslant k_1$, $\mathit{maxgaps}(c) \leqslant k_2$ and $\mathit{maxc}^+(c) \leqslant k_3$ is NP-complete if $k_1 \geqslant 2$ and $k_2 \geqslant 1$ and $k_3 \geqslant 1$, and it is in P if $k_1 \leqslant 1$ or $k_2 = 0$ or $k_3 = 0$.*

Proof sketch. The theorem's second statement is obtained by simple algorithms. We sketch a hardness reduction that proves the first statement with an example. Let (x_1, x_2, x_4), (x_4, x_5, x_7), and (x_1, x_3, x_5) be the clauses of a Boolean formula in 3-CNF. We construct $s := y_1 x_1 x_2 x_4 y_2\ z\ y_3 x_4 x_5 x_7 y_4\ z\ y_5 x_1 x_3 x_5 y_6$ (where all x_i, y_i and z are variables) and $t := 0001000\ 1\ 0001000\ 1\ 0001000$ (where $\Gamma = \{0, 1\}$). It can be verified that if $\mu(s) \preccurlyeq_e t$ with $\mu(z) = 1$ and e is such that all gaps are 0 or 1, except the gaps between x_i variables, which are 0, then $\mu(x_i) = 1$ for exactly one variable in each clause (i. e., the Boolean formula is "1-in-3 satisfiable"). In order to force $\mu(z) = 1$, we have to add a sufficient number of occurrences of z and 1 as prefixes to s and t, respectively. ◀

Our results discussed so far show that for every parameter $p \in \{|t|, w, |s|, \mathit{repvars}(s)\}$, the problem $\mathcal{Q}$-Match can be solved in polynomial time if p is bounded by a constant. However, the degree of the polynomial may depend on p, which even for small p may lead to prohibitively high running times. Therefore, we ask next for which parameters p (or parameter combinations $(p_1, \dots, p_r)$), we can achieve more attractive running times of the form $f(p) \cdot \mathrm{poly}(|q|, |t|)$, for some computable function f. More precisely, we investigate the parameterised complexity of $\mathcal{Q}$-Match by asking the following questions:

▶ **Question 4.4.** *Is $\mathcal{Q}$-Match parameterised by $(p_1, p_2, \dots, p_r)$ in FPT (i. e., can it be solved in time $\mathrm{O}(f(p_1, p_2, \dots, p_r) \cdot g(|s|, |t|))$ for some polynomial g and a computable function f), or is this not the case (subject to common complexity theoretical assumptions)?*

An answer for parameters $(|\mathit{types}(t)|, \mathit{maxgaps}(c), \mathit{maxc}^+(c))$ follows from Theorem 4.3:

▶ **Corollary 4.5.** *$\mathcal{Q}$-Match parameterised by $(|\mathit{types}(t)|, \mathit{maxgaps}(c), \mathit{maxc}^+(c))$ is not in FPT (unless P = NP).*

From the brute-force approaches and the (more or less) obvious dependencies between the parameters, we can conclude fixed-parameter tractability for several parameterisations:

▶ **Theorem 4.6.** *$\mathcal{Q}$-Match is in FPT with respect to any of the following parameterisations: $(|\mathit{types}(t)|, |\mathit{repvars}(s)|)$; $(|s|, \mathit{maxgaps}(c))$; $(|t|)$; (w); $(|s|, \mathit{maxc}^+(c))$; $(|s|, \mathit{types}(t))$.*

Next, we show that all remaining parameterisations that are not covered by Theorem 4.6 or Corollary 4.5 yield W[1]-hard variants, which are therefore not in FPT (modulo the

common complexity theoretical assumption that W[1] $\neq$ FPT). We assume that the reader is familiar with basic concepts of parameterised complexity theory (cf., e.g., [12, 17, 13]).

▶ **Theorem 4.7.** *Each of the following two problems is* W[1]*-hard:*
- $\mathcal{Q}$-Match *parameterised by* $(|repvars(s)|, maxgaps(c) \leqslant 1, maxc^{+}(c) \leqslant 1)$,
- $\mathcal{Q}$-Match *parameterised by* $(|s|, |repvars(s)|)$.

Proof sketch of Theorem 4.7. We devise suitable reductions from the following problem, which is known to be W[1]-hard with respect to parameter k (cf., [12]):

Multi-Colored Clique (MCClique). The input is a graph $G = (V, E)$ with a partition $V_1, V_2, \ldots, V_k$ of V such that each V_i is an independent set. The question is whether G has a clique of size k (note that such a clique has to contain exactly one vertex from each V_i).

Let $G = (V, E)$ be a graph with a partition $V_1, V_2, \ldots, V_k$ of V such that every V_i is an independent set. Let $n := |V|$, $p_i := |V_i|$ and $V_i = \{v_1^i, v_2^i, \ldots, v_{p_i}^i\}$ for all $i \in [k]$. We have to construct a trace t and an swg-query $q = (s, w, c)$ such that $t \models q$ iff G has a clique of size k. For constructing the trace t, we use a distinct type for every $v \in V$, and we also use two special types \$ and $\diamond$. The query string s will be built using distinct variables $x_1, x_2, \ldots, x_k$, $z_\$$, $y_1, \ldots, y_r$ where r is a suitably chosen number of size polynomial in n and k (the precise choice of r will become clear below, from the construction of s and t). The variables $x_1, x_2, \ldots, x_k$ correspond to the k vertices of the clique, and the variable $z_\$$ will be used as a separator. The variables $x_1, x_2, \ldots, x_k, z_\$$ may be repeated variables of s, while each of the variables $y_1, \ldots, y_r$ will occur in s only once. To avoid notational clutter, we will denote all occurrences of variables y_i simply by y and keep in mind that each occurrence of y is actually an occurrence of an individual variable from $\{y_i \ : \ i \in [r]\}$. Our choice of s and t relies on some gadgets, which we discuss next. The gadget s_V lists all variables x_i separated by $y^n z_\$ y^n$, and t_V lists the vertices of the sets V_i separated by $\diamond^n \$ \diamond^n$. More formally,

$$\begin{array}{lccccccccccc} s_V := & y^n & x_1 & & y^n\ z_\$\ y^n & x_2 & & y^n\ z_\$\ y^n & \ldots & y^n\ z_\$\ y^n & x_k & & y^n \\ t_V := & \diamond^n & v_1^1 \ldots v_{p_1}^1 & & \diamond^n\ \$\ \diamond^n & v_1^2 \ldots v_{p_2}^2 & & \diamond^n\ \$\ \diamond^n & \ldots & \diamond^n\ \$\ \diamond^n & v_1^k \ldots v_{p_k}^k & & \diamond^n \end{array}$$

The purpose of this gadget is as follows: We can show that if $\mu(s_V) \preccurlyeq t_V$ for some $\mu : \mathsf{Vars} \to \Gamma$ with $\mu(z_\$) = \$$, then $(\mu(x_1), \mu(x_2), \ldots, \mu(x_k)) \in V_1 \times V_2 \times \cdots \times V_k$, and there exists an embedding e with *maximum gap size* 1 (i.e., $\max\{e(i{+}1){-}1 - e(i) \colon i \in [|s_V|{-}1]\} \leqslant 1$) and a substitution μ' that differs from μ only on variables in $\{y_1, \ldots, y_r\}$, such that $\mu'(s_V) \preccurlyeq_e t_V$. Note that for ensuring the latter property, we need the occurrences of y^n.

In order to enforce that the set $\{\mu(x_1), \mu(x_2), \ldots, \mu(x_k)\}$ is a clique, we use the following gadgets. For every i, j with $1 \leqslant i < j \leqslant k$, let $(u_1^i, u_1^j), \ldots, (u_{q_{i,j}}^i, u_{q_{i,j}}^j)$ be a list of exactly the edges between V_i and V_j, and let this list be ordered lexicographically. We let

$$s_{i,j} := \ y^{3n^2}\ (x_i x_j)^3\ y^{3n^2} \quad \text{and} \quad t_{i,j} := \ \diamond^{3n^2} (u_1^i u_1^j)^3 (u_2^i u_2^j)^3 \ldots (u_{q_{i,j}}^i u_{q_{i,j}}^j)^3\ \diamond^{3n^2} \ .$$

The purpose of this gadget is as follows: We can show that if there is a substitution μ with $\mu(x_i) \in V_i$, $\mu(x_j) \in V_j$ and $\mu(s_{i,j}) \preccurlyeq t_{i,j}$, then $(\mu(x_i), \mu(x_j)) \in E$ and there exists an embedding e with maximum gap size 1 and a substitution μ' that differs from μ only on variables in $\{y_1, \ldots, y_r\}$ such that $\mu'(s_{i,j}) \preccurlyeq_e t_{i,j}$. In particular, this gadget enforces that, for every $1 \leqslant i < j \leqslant k$, μ maps x_i and x_j to *adjacent* vertices from V_i and V_j (for this, we essentially use that the list of edges in $t_{i,j}$ is ordered lexicographically).

Finally, we combine all the gadgets into the query string s and trace t as follows:

$$\begin{array}{lll} s & := & (z_\$)^{3n^2+1} s_V\ z_\$\ s_{1,2}\ z_\$\ s_{1,3}\ z_\$\ \cdots\ z_\$\ s_{1,k}\ z_\$\ s_{2,3}\ z_\$\ \cdots\ z_\$\ s_{2,k}\ z_\$\ \cdots\ z_\$\ s_{k-1,k} \\ t & := & (\$)^{3n^2+1}\ t_V\ \$\ t_{1,2}\ \$\ t_{1,3}\ \$\ \cdots\ \$\ t_{1,k}\ \$\ t_{2,3}\ \$\ \cdots\ \$\ t_{2,k}\ \$\ \cdots\ \$\ t_{k-1,k} \end{array}$$

Table 1 An integer entry means that the parameter is bounded by this constant, the entry "**p**" means that the problem is parameterised by this parameter, and the entry "–" means that the parameter is not bounded by a constant and the problem is not parameterised by this parameter.

$\|t\|$	w	$\|s\|$	$\|types(t)\|$	$\|repvars(s)\|$	$maxgaps(c)$	$maxc^+(c)$	Complexity
p	–	–	–	–	–	–	FPT (Thm. 4.6)
–	**p**	–	–	–	–	–	FPT (Thm. 4.6)
–	–	**p**	–	–	–	**p**	FPT (Thm. 4.6)
–	–	**p**	**p**	–	–	–	FPT (Thm. 4.6)
–	–	**p**	–	**p**	–	–	W[1]-hard (Thm. 4.7)
–	–	**p**	–	–	**p**	–	FPT (Thm. 4.6)
–	–	–	**p**	**p**	–	–	FPT (Thm. 4.6)
–	–	–	2	–	1	1	NP-hard (Thm. 4.3)
–	–	–	–	**p**	1	1	W[1]-hard (Thm. 4.7)

The construction of the query $q = (s, w, c)$ is completed by choosing $w = \infty$ (or $w = |t|$, which does not make any difference here), and $(c_i^-, c_i^+) = (0, 0)$ for every $1 \leqslant i \leqslant 3n^2$, and $(c_i^-, c_i^+) = (0, 1)$ for every $3n^2{+}1 \leqslant i \leqslant |s|{-}1$.

It can now be verified that $t \models q$ if and only if G contains a clique of size k. This completes the proof sketch of the theorem's first statement.

For proving the second statement, we modify the reduction such that we use constant length separators y and $\diamond$ instead of y^n, $\diamond^n$, y^{3n^2} and $\diamond^{3n^2}$. This modification allows us to use prefixes $(z_\$)^2$ and $(\$)^2$ instead of $(z_\$)^{3n^2+1}$ and $(\$)^{3n^2+1}$, which means that the total length of $|s|$ only depends on k (but now neither $maxgaps(c)$ nor $maxc^+(s)$ are bounded anymore). This yields the theorem's second statement. ◀

We note that the theorems from above answer Question 4.4 for every subset $p_1, p_2, \ldots, p_r$ of our parameters – this can be verified with the help of Table 1.

We conclude this section by discussing some related questions. Query strings can also be considered as sequences of distinct variables $x_1 x_2 \cdots x_n$ enriched with constraints "$x_i = x_j$" and "$x_i = \gamma$" for $\gamma \in \Gamma$. If the first type of constraint is not used, then $|repvars(s)| = 0$, and therefore the matching problem is in P (Theorem 4.2). If, instead, we disallow constraints "$x_i = \gamma$" (i. e., query strings must not contain types), then this does not make the matching problem any easier: all our reductions produce query strings that consist of variables only. Giving up the global window size is equivalent to setting it to ∞, which means that the lower bounds of Theorems 4.3 and 4.6 still apply (note that the reduction sets the global window size to ∞). On the other hand, for swg-queries without local gap-size constraints (i. e., with local gap-size constraints of the form $((0, \infty), (0, \infty), \ldots, (0, \infty))$) the complexity of the matching problem is open. We plan to address this in the full version of this paper.

In order to use the algorithm of Section 3, we need to fix the length ℓ of the query string, the global window size w, and the tuple c of local gap-size constraints. Finding suitable values for ℓ and c seems comparatively simple: firstly, we can afford to try out several combinations (e. g., all combinations of values $\ell \in \{5, 6, \ldots, 10\}$ and $w \in \{100, 1000, 5000\}$, which requires 18 runs of the algorithm), and, secondly, it seems likely that some a-priori knowledge of the data will lead to reasonable choices for these parameters. The tuple of local gap-size constraints, on the other hand, is a more complex parameter and, especially if we want to try the algorithm on different query string sizes, it seems likely that the best we can do is to give a reasonable gap-size constraint for *all* gaps, i. e., local gap-size constraints $((i, j), (i, j), \ldots, (i, j))$. Whether the matching problem for such swg-queries is also NP-hard is not answered by the reduction of Theorem 4.3, since it uses different types of gap-size

constraints. However, the reduction can be easily adapted: Instead of the factors $y_1x_1x_2x_4y_2$ of the query string, we use factors $y_1y_2y_3(x_1)^2(x_2)^2(x_4)^2y_4y_5y_6$, and instead of the factors 0001000 of the trace, we use factors 0^7110^7. It can be verified that if $\mu(s) \preccurlyeq_e t$ with $\mu(z) = 1$ and e is such that *all gaps* are 0 or 1, then $\mu(x_i) = 1$ for exactly one variable in each clause.

5 Practical Considerations and Concluding Remarks

Motivated by the task of discovering patterns of interest in event streams, we introduced swg-queries (Section 2), we formalised query discovery as computing descriptive queries (Section 3), we presented an algorithm for this task (Algorithm 1), and we provided an in-depth complexity analysis of the problem of query discovery (Section 4).

What are possible practical implications of our theoretical study? To approach this question, let us briefly discuss how the complexity bounds of Section 4 can help to manage our expectations. Let $\|(\ell, w, c)\|$ and $\|\mathcal{S}\|$ be the size of reasonable encodings of (ℓ, w, c) and $\mathcal{S}$, respectively. In a practical scenario, we can expect $\|\mathcal{S}\|$ (and therefore possibly also $\|w\|$ and $\|c\|$, see Remark 3.3) to be large, but we can assume a moderate query string length ℓ, since queries should still be understandable by users. Hence, running times $\mathrm{O}(f(\ell)g(\|\mathcal{S}\|, \|(\ell, w, c)\|))$ might be practically relevant even for exponential function f and low-degree polynomial g. Unfortunately, Theorem 4.7 and the reduction of Theorem 3.6 rule out such algorithms. On the other hand, Theorems 4.6 and 3.5 show that we can indeed solve $\mathcal{Q}^{|rv(s)|\leqslant k}$-CompDescQuery in time $\mathrm{O}(f(k, |types(\mathcal{S})|)g(\|\mathcal{S}\|, \|(\ell, w, c)\|))$ for polynomial g, where $\mathcal{Q}^{|rv(s)|\leqslant k}$ is the class of queries with $|repvars(q)| \leqslant k$. This might be of practical interest, since our examples discussed in Section 1 suggest that $|repvars(q)|$ and $|types(\mathcal{S})|$ are small. In fact, even a restriction like, say $|repvars(q)| \leqslant 7$ and $|types(\mathcal{S})| \leqslant 15$, seems still practically relevant, and in this case our algorithm has a low-degree polynomial running time.

These theoretical considerations justify hope that, despite the inherent hardness of query discovery, our algorithm is also practically worthwhile. As a proof of concept and in order to investigate our algorithms' performance on real world data, we implemented a prototype in Python and conducted experiments on Google Cluster Traces [26]. We summarise our experiments' main results as follows:

- We discovered queries from the Google Cluster Traces, including queries that contain situations of interest. However, we also discovered queries that do not contain real situations of interest (but still are descriptive according to Definition 3.1).
- Considering different query string lengths or support thresholds, we observed expected correlations concerning the overall runtime and the number of queries q' for which the support $\mathsf{supp}(q', \mathcal{S})$ is computed.
- Comparing datasets regarding runtime (or the number of queries q' for which the support $\mathsf{supp}(q', \mathcal{S})$ is computed), for a fixed query string length, it turns out that they do not only depend on the dataset size, but also on the order in which the positions of the query string are considered, and on whether variable operations have to be tested or not.

As future work, we plan to further develop our prototype implementation and extend our initial experiments to a comprehensive experimental analysis. In particular, we are interested in evaluating heuristics and to explore possible ways for improving the algorithm's practical performance. For example, treating the check of $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$ as $|\mathcal{S}|$ independent instances of the matching problem makes sense for our theoretical considerations, but in practice it might be possible to exploit the fact that q' does not change for these $|\mathcal{S}|$ instances, or that in each run of the algorithm $\mathcal{S}$ is the same for *all* checks of $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$, or that whenever we check $\mathsf{supp}(q', \mathcal{S}) \geqslant \mathsf{sp}$, we already checked $\mathsf{supp}(q'', \mathcal{S}) \geqslant \mathsf{sp}$ for a rather similar query q''.

References

1 Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994. URL: http://www.vldb.org/conf/1994/P487.PDF.

2 Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 3–14. IEEE Computer Society, 1995. doi:10.1109/ICDE.1995.380415.

3 Dana Angluin. Inductive inference of formal languages from positive data. *Inf. Control.*, 45(2):117–135, 1980. doi:10.1016/S0019-9958(80)90285-5.

4 Alexander Artikis, Chris Baber, Pedro Bizarro, Carlos Canudas-de-Wit, Opher Etzion, Fabiana Fournier, Paul Goulart, Andrew Howes, John Lygeros, Georgios Paliouras, Assaf Schuster, and Izchak Sharfman. Scalable proactive event-driven decision making. *IEEE Technol. Soc. Mag.*, 33(3):35–41, 2014. doi:10.1109/MTS.2014.2345131.

5 Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. Complex event recognition languages: Tutorial. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*, pages 7–10. ACM, 2017. doi:10.1145/3093742.3095106.

6 Alexander Artikis, Matthias Weidlich, François Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann, Katharina Morik, Vana Kalogeraki, Jakub Marecek, Avigdor Gal, Shie Mannor, Dimitrios Gunopulos, and Dermot Kinane. Heterogeneous stream processing and crowdsourcing for urban traffic management. In Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy, editors, *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*, pages 712–723. OpenProceedings.org, 2014. doi:10.5441/002/edbt.2014.77.

7 Lars Baumgärtner, Christian Strack, Bastian Hoßbach, Marc Seidemann, Bernhard Seeger, and Bernd Freisleben. Complex event processing for reactive security monitoring in virtualized computer systems. In Frank Eliassen and Roman Vitenberg, editors, *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, Oslo, Norway, June 29 - July 3, 2015*, pages 22–33. ACM, 2015. doi:10.1145/2675743.2771829.

8 Lei Chang, Tengjiao Wang, Dongqing Yang, and Hua Luan. Seqstream: Mining closed sequential patterns over stream sliding windows. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 83–92. IEEE Computer Society, 2008. doi:10.1109/ICDM.2008.36.

9 Hong Cheng, Xifeng Yan, and Jiawei Han. Incspan: incremental mining of sequential patterns in large database. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 527–532. ACM, 2004. doi:10.1145/1014052.1014114.

10 Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012. doi:10.1145/2187671.2187677.

11 Joel D. Day, Pamela Fleischmann, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. The edit distance to k-subsequence universality. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, pages 25:1–25:19, 2021. doi:10.4230/LIPIcs.STACS.2021.25.

12 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

14 Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. Revisiting Shinohara's algorithm for computing descriptive patterns. *Theor. Comput. Sci.*, 733:44–54, 2018. doi:10.1016/j.tcs.2018.04.035.

15 Henning Fernau and Markus L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Inf. Comput.*, 242:287–305, 2015. doi:10.1016/j.ic.2015.03.006.

16 Henning Fernau, Markus L. Schmid, and Yngve Villanger. On the parameterised complexity of string morphism problems. *Theory Comput. Syst.*, 59(1):24–51, 2016. doi:10.1007/s00224-015-9635-3.

17 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

18 Dominik D. Freydenberger and Daniel Reidenbach. Existence and nonexistence of descriptive patterns. *Theor. Comput. Sci.*, 411(34-36):3274–3286, 2010. doi:10.1016/j.tcs.2010.05.033.

19 Dominik D. Freydenberger and Daniel Reidenbach. Inferring descriptive generalisations of formal languages. *J. Comput. Syst. Sci.*, 79(5):622–639, 2013. doi:10.1016/j.jcss.2012.10.001.

20 Pawel Gawrychowski, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. Efficiently testing Simon's congruence. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, pages 34:1–34:18, 2021. doi:10.4230/LIPIcs.STACS.2021.34.

21 Lars George, Bruno Cadonna, and Matthias Weidlich. Il-miner: Instance-level discovery of complex event patterns. *Proc. VLDB Endow.*, 10(1):25–36, 2016. doi:10.14778/3015270.3015273.

22 Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020. doi:10.1007/s00778-019-00557-w.

23 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.

24 Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. Learning from the past: automated rule generation for complex event processing. In Umesh Bellur and Ravi Kothari, editors, *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014*, pages 47–58. ACM, 2014. doi:10.1145/2611286.2611289.

25 A. Mateescu and A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 230–242. Springer, 1997.

26 Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, pages 1–14, 2011.

27 T. Shinohara. Polynomial time inference of pattern languages and its application. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 191–209, 1982.

28 Takeshi Shinohara and Setsuo Arikawa. Pattern inference. In *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*, pages 259–291, 1995. doi:10.1007/3-540-60217-8_13.

29 Kia Teymourian, Malte Rohde, and Adrian Paschke. Knowledge-based processing of complex stock market events. In Elke A. Rundensteiner, Volker Markl, Ioana Manolescu, Sihem Amer-Yahia, Felix Naumann, and Ismail Ari, editors, *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, pages 594–597. ACM, 2012. doi:10.1145/2247596.2247674.

30 Jianyong Wang and Jiawei Han. BIDE: efficient mining of frequent closed sequences. In Z. Meral Özsoyoglu and Stanley B. Zdonik, editors, *Proceedings of the 20th International*

Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA, pages 79–90. IEEE Computer Society, 2004. doi:10.1109/ICDE.2004.1319986.

31 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 217–228. ACM, 2014. doi:10.1145/2588555.2593671.

Streaming Enumeration on Nested Documents

Martín Muñoz
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile

Abstract

Some of the most relevant document schemas used online, such as XML and JSON, have a nested format. In the last decade, the task of extracting data from nested documents over streams has become especially relevant. We focus on the streaming evaluation of queries with outputs of varied sizes over nested documents. We model queries of this kind as Visibly Pushdown Transducers (VPT), a computational model that extends visibly pushdown automata with outputs and has the same expressive power as MSO over nested documents. Since processing a document through a VPT can generate a massive number of results, we are interested in reading the input in a streaming fashion and enumerating the outputs one after another as efficiently as possible, namely, with constant-delay. This paper presents an algorithm that enumerates these elements with constant-delay after processing the document stream in a single pass. Furthermore, we show that this algorithm is worst-case optimal in terms of update-time per symbol and memory usage.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Streaming, nested documents, query evaluation, enumeration algorithms

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.19

Related Version *Extended Version*: `https://arxiv.org/abs/2010.06037`

Funding This work was funded by ANID - Millennium Science Initiative Program - Code `ICN17_002`.

1 Introduction

Streaming query evaluation [2, 9] is the task of processing queries over data streams in one pass and with a limited amount of resources. This approach is especially useful on the web, where servers share data, and they have to extract the relevant content as they receive it. For structuring the data, the de facto structure on the web are nested documents, like XML or JSON. For querying, servers use languages designed for these purposes, like XPath, XQuery, or JSON query languages. As an illustrative example, suppose our data server (e.g. Web API) is continuously receiving XML documents of the form:

```
<doc> <a> <b/> <c/> <b/> </a> <c> <b/> <b/> </c> </doc> ...
```

and for each document it has to evaluate the query $\mathcal{Q} = //a/b$ (i.e., to extract all b-tags that are surrounded by an a-tag). The streaming query evaluation problem consists on reading these documents and finding all b-tags without storing the entire document on memory, i.e., by making one pass over the data and spending constant time per tag. In our example, we need to retrieve the 3rd and 5th tag as soon as the last tag `</doc>` is received. One could consider here that the server has to read an infinite stream and perform the query evaluation continuously, where it must enumerate partial outputs as one of the XML documents ends.

Researchers have studied the streaming query evaluation problem in the past, focusing on reducing the processing time or memory usage (see, e.g. [13]). Hence, they spent less effort on understanding the enumeration time of such a problem, regarding delay guarantees

between outputs. Constant-delay enumeration is a new notion of efficiency for retrieving outputs [23, 44]. Given an instance of the problem, an algorithm with constant-delay enumeration performs a preprocessing phase over the instance to build some indices and then continues with an enumeration phase. It retrieves each output, one-by-one, taking a delay that is constant between any two consecutive outcomes. These algorithms provide a strong guarantee of efficiency since a user knows that, after the preprocessing phase, she will access the output as if the algorithm had already computed it. These techniques have attracted researchers' attention, finding sophisticated solutions to several query evaluation problems [11, 15, 10, 5, 26, 6].

In this work, we investigate the streaming query evaluation problem over nested documents by including enumeration guarantees, like constant-delay. We study the evaluation of queries given by Visibly Pushdown Transducers (VPT) over nested documents. These machines are the natural "output extension" of visibly pushdown automata, and have the same expressive power as MSO over nested documents. In particular, VPT can define queries like $\mathcal{Q}$ above or any fragment of query languages for XML or JSON included in MSO. Therefore, VPT allow considering the streaming query evaluation from a more general perspective, without getting married to a specific language (e.g., XPath).

We study the evaluation of VPT over a nested document in a streaming fashion. Specifically, we want to find a streaming algorithm that reads the document sequentially and spends constant time per input symbol. Furthermore, whenever needed, the algorithm can enumerate all outputs with output-linear delay. The main contribution of the paper is an algorithm with such characteristics for the class of I/O-unambiguous VPT. We can extend this algorithm by determinization to all VPT (i.e., in data complexity). Regarding memory consumption, we bound the amount of memory used in terms of the nesting of the document and the output weight. We show that our algorithm is worst-case optimal in the sense that there are instances where the maximum amount of memory required by any streaming algorithm is at least one of these two measures.

Related work. The problem of streaming query evaluation has been extensively studied in the last decades. Some work considered streaming verification, like schema validation [45] or type-checking [38], where the output is true or false. Other proposals [19, 42, 36, 31, 41] provided streaming algorithms for XPath or XQuery's fragments; however, extending them for reaching constant-delay enumeration seems unlikely. Furthermore, most of these works [38, 30, 29] assumed outputs of fixed size (i.e., tuples). People have also considered other aspects of streaming evaluation with outputs like earliest query answering [29] or bounded delay [28] (i.e., given the first visit of a node, find the earliest event that permits its selection). These aspects are orthogonal to the problem studied here. Another line of research is [12, 13], which presents space lower bounds for evaluating fragments of XPath or XQuery over streams. These works do not consider restrictions on the delay to give outputs.

Visibly pushdown automata [4] are a model usually used for streaming evaluation of boolean queries [38]. In [24, 3], authors studied the evaluation of VPT in a streaming fashion, but none of them saw enumeration problems. Other extensions [27] for streaming evaluation have been analyzed but restricted to fixed-size outputs, and constant-delay was not included.

Constant-delay algorithms have been studied for several classes of query languages and structures [44], as we already discussed. In [10, 5], researchers considered query evaluation over trees (i.e., a different representation for nested documents), but their algorithms are for offline evaluation and it is not clear how to extend this algorithm for the online setting. This research is extended with updates in [7], which can encode streams by inserting new

data items to the left. However, their update-time is logarithmic, and our proposal can do it with constant time (i.e., in data complexity). Furthermore, to the best of our knowledge it is unclear how to modify the work in [7] to get constant update-time in our scenario. Streaming evaluation with constant-delay enumeration was included in the context of dynamic query evaluation [34, 16, 40, 37] or complex event processing [33, 32]. In both cases, the input cannot encode nested documents, and their results do not apply.

2 Preliminaries

Well-nested words and streams. As usual, given a set Σ we denote by Σ^* all finite words with symbols in Σ where $\varepsilon \in \Sigma^*$ represents the empty word of length 0.

We will work over a *structured alphabet* $\Sigma = (\Sigma^<, \Sigma^>, \Sigma^|)$ comprised of three disjoint sets $\Sigma^<$, $\Sigma^>$, and $\Sigma^|$ that contain *open*, *close*, and *neutral* symbols respectively (in [4, 25] these sets are named *call*, *return*, and *local*, respectively). Furthermore, we will denote symbols in $\Sigma^<$, $\Sigma^>$ or $\Sigma^|$ by $\langle a$, $a\rangle$, and a, respectively. Instead, we will use s to denote any symbol in $\Sigma^<$, $\Sigma^>$, or $\Sigma^|$. The set of *well-nested words* over Σ, denoted as $\Sigma^{<*>}$, is defined as the closure of the following rules: $\Sigma^| \cup \{\varepsilon\} \subseteq \Sigma^{<*>}$, if $w_1, w_2 \in \Sigma^{<*>} \setminus \{\varepsilon\}$ then $w_1 \cdot w_2 \in \Sigma^{<*>}$, and if $w \in \Sigma^{<*>}$ and $\langle a \in \Sigma^<$ and $b\rangle \in \Sigma^>$ then $\langle a \cdot w \cdot b\rangle \in \Sigma^{<*>}$. In addition, we will work with prefixes of well-nested words, that we call *prefix-nested words*. We denote the set of prefixes of $\Sigma^{<*>}$ as $\mathsf{prefix}(\Sigma^{<*>})$. Also, we will sometimes use $w[i]$ to refer to the i-th symbol in a word w.

A *stream* $\mathcal{S} = s_1 s_2 \cdots$ is an infinite sequence where $s_i \in \Sigma^< \cup \Sigma^> \cup \Sigma^|$. Given a stream $\mathcal{S} = s_1 s_2 \ldots$ and positions $i, j \in \mathbb{N}$ such that $i \leq j$, the word $\mathcal{S}[i, j]$ is the sequence $s_i \cdots s_j$. We also use this notation to refer to subsequences of infinite sequences that are not composed of symbols in Σ. For a stream $\mathcal{S}$, we will always assume that for each $i \in \mathbb{N}$, the word $\mathcal{S}[1, i]$ is a prefix of some nested word (i.e., it can be completed to form a nested word). We also consider a method $\texttt{yield}[\mathcal{S}]$ which can be called to access each element of $\mathcal{S}$ sequentially.

Visibly pushdown automata. A *visibly pushdown automaton* [4] (VPA) is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, I, F)$ where Q is a finite set of states, $\Sigma = (\Sigma^<, \Sigma^>, \Sigma^|)$ is the input alphabet, Γ is the stack alphabet, $\Delta \subseteq (Q \times \Sigma^< \times Q \times \Gamma) \cup (Q \times \Sigma^> \times \Gamma \times Q) \cup (Q \times \Sigma^| \times Q)$ is the transition relation, $I \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of final states. A transition $(q, \langle a, q', \gamma)$ is a *push-transition* where on reading $\langle a \in \Sigma^<$, γ is pushed onto the stack and the current state switches from q to q'. Conversely, $(q, a\rangle, \gamma, q')$ is a *pop-transition* where on reading $a\rangle \in \Sigma^>$ from the input and γ from the top of the stack, the current state changes from q to q', and the symbol γ is popped. Lastly, we say that (q, a, q') is a *neutral transition* if $a \in \Sigma^|$, where there is no stack operation.

A stack is a finite sequence σ over Γ where the top of the stack is the first symbol on σ. For a well-nested word $w = s_1 \cdots s_n$ in $\Sigma^{<*>}$, a run of $\mathcal{A}$ on w is a sequence $\rho = (q_1, \sigma_1) \xrightarrow{s_1} \ldots \xrightarrow{s_n} (q_{n+1}, \sigma_{n+1})$, where each $q_i \in Q$, $\sigma_i \in \Gamma^*$, $q_1 \in I$, $\sigma_1 = \varepsilon$, and for every $i \in [1, n]$ the following holds: (1) if $s_i \in \Sigma^<$, then there is $\gamma \in \Gamma$ such that $(q_i, s_i, q_{i+1}, \gamma) \in \Delta$ and $\sigma_{i+1} = \gamma\sigma_i$, (2) if $s_i \in \Sigma^>$, then there is $\gamma \in \Gamma$ such that $(q_i, s_i, \gamma, q_{i+1}) \in \Delta$ and $\sigma_i = \gamma\sigma_{i+1}$, and (3) if $s_i \in \Sigma^|$, then $(q_i, s_i, q_{i+1}) \in \Delta$ and $\sigma_{i+1} = \sigma_i$. A run ρ is accepting if $q_{n+1} \in F$. A well-nested word $w \in \Sigma^{<*>}$ is accepted by a VPA $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The language $\mathcal{L}(\mathcal{A})$ is the set of well-nested words accepted by $\mathcal{A}$. Note that if ρ is an accepting run of $\mathcal{A}$ on a well-nested word w, then $\sigma_{n+1} = \varepsilon$. A set of well-nested words $\mathcal{L} \subseteq \Sigma^{<*>}$ is called a visibly pushdown language if there exists a VPA $\mathcal{A}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$.

A VPA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, I, F)$ is said to be *deterministic* if $|I| = 1$ and δ is a function subset of $(Q \times \Sigma^{\langle} \to Q \times \Gamma) \cup (Q \times \Sigma^{\rangle} \times \Gamma \to Q) \cup (Q \times \Sigma^{|} \to Q)$. We also say that $\mathcal{A}$ is *unambiguous* if, for every $w \in \mathcal{L}(\mathcal{A})$, there exists exactly one accepting run of $\mathcal{A}$ on w. In [4], it is shown that for every VPA there exists an equivalent deterministic VPA of at most exponential size.

Model of computation. As it is common in the enumeration algorithms literature [10, 21, 44], for our algorithms we assume the computational model of Random Access Machines (RAM) with uniform cost measure, and addition and subtraction as basic operations [1]. We assume that a RAM has read-only input registers where the machine places the input, read-write work registers where it does the computation, and write-only output registers where it gives the output (i.e., the enumeration of the results).

3 Streaming evaluation with output-linear delay

We are interested in defining a notion of a streaming enumeration problem: to evaluate a query over a stream and to enumerate the outputs with bounded delay whenever there is such. Towards this goal, we want to restrict the amount of resources used (i.e., time and space) and impose strong guarantees on the delay. As our gold standard, we consider the notion of *output-linear delay* defined in [26]. This notion is a refinement of the definition of constant-delay [44] or linear-delay [21] enumeration that better fits our purpose. Altogether, our plan for this section is to define a streaming enumeration problem and then provide a notion of efficiency that a solution for this problem should satisfy.

We adopt the setting of relations to formalize a streaming enumeration problem [35, 8]. First, we need to define what is an enumeration problem outside the stream setting. Let Ω be an alphabet. An enumeration problem is a relation $R \subseteq (\Omega^* \times \Omega^*) \times \Omega^*$. For each pair $((q, x), y) \in R$ we view (q, x) as the input of the problem and y as a possible output for (q, x). Furthermore, we call q the query and x the data. This separation allows for a fine-grained analysis of the query complexity and data complexity of the problem. For an instance (q, x) we define the set $[\![q]\!]_R(x) = \{y \mid ((q, x), y) \in R\}$ of all outputs of evaluating q over x.

A streaming enumeration problem is an extension of an enumeration problem R where the input is a pair $(q, \mathcal{S})$ such that $\mathcal{S}$ is an infinite sequence of elements in Ω. We identify two ways of extending an enumeration problem R that differ in the output sets that are desired at each position in the stream:

1. The *streaming full-enumeration problem* for R is one where the objective is to enumerate the set $[\![q]\!]_R(\mathcal{S}[1, n])$ at each position $n \geq 1$.
2. A *streaming Δ-enumeration problem* for R is one where the objective is to enumerate the set $[\![q]\!]^{\Delta}_R(\mathcal{S}[1, n]) = [\![q]\!]_R(\mathcal{S}[1, n]) \setminus \bigcup_{i<n} [\![q]\!]_R(\mathcal{S}[1, i])$ at each position $n \geq 1$.

These versions give us two different ways of returning the outputs. These notions have been studied previously in the context of incremental view maintenance [20] and more recently, for dynamic query evaluation [34, 16]. For the sake of simplification, in the following we provide all definitions for the full-enumeration scenario. All definitions can be extended to Δ-enumeration by changing $[\![q]\!]_R$ to $[\![q]\!]^{\Delta}_R$.

We turn now to our notion of efficiency for solving a streaming enumeration problem. Let $f \colon \mathbb{N} \to \mathbb{N}$. We say that $\mathcal{E}$ is a *streaming evaluation algorithm* for R with *f-update-time* if $\mathcal{E}$ operates in the following way: it receives a query q and reads the stream $\mathcal{S}$ by calling the `yield`$[\mathcal{S}]$ method sequentially. After the n-th call to `yield`$[\mathcal{S}]$, the algorithm processes the n-th data symbol in two phases:

- In the first phase, called the *update* phase, the algorithm updates a data structure D with the read symbol and the time spent is bounded by $\mathcal{O}(f(|q|))$.
- The second phase, called the *enumeration* phase, occurs immediately after each update phase and outputs $[\![q]\!]_R(\mathcal{S}[1,n])$ using D. During this phase the algorithm: (1) writes $\#y_1\#y_2\#\cdots\#y_m\#$ to the output registers where $\#$ is a distinct separator symbol not contained in Ω, and $y_1, y_2, \ldots, y_m$ is an enumeration (without repetitions) of the set $[\![q]\!]_R(\mathcal{S}[1,n])$, (2) it writes the first $\#$ as soon as the enumeration phase starts, and (3) it stops immediately after writing the last $\#$.

The purpose of separating $\mathcal{E}$'s operation into an update and enumeration phase is to make an output-sensitive analysis of $\mathcal{E}$'s complexity. Moreover, from a user perspective, this separation allows running the enumeration phase without interrupting the update phase. That is, the user could execute the enumeration phase in a separate machine, and its running time only depends on how many outputs she wants to enumerate.

For the enumeration phase, we measure the delay between two outputs as follows: For an input $x \in \Omega^*$, let $\#y_1\#y_2\#\cdots\#y_m\#$ be the output of the algorithm during any call to the enumeration phase. Furthermore, let $\mathsf{time}_i(x)$ be the time in the enumeration phase when the algorithm writes the i-th $\#$ when running on x for $i \leq m+1$. Define $\mathsf{delay}_i(x) = \mathsf{time}_{i+1}(x) - \mathsf{time}_i(x)$ for $i \leq m$. Then we say that $\mathcal{E}$ has *output-linear delay* if there exists a constant k such that for every $x \in \Omega^*$ and $i \leq m$ it holds that $\mathsf{delay}_i(x) \leq k \cdot |y_i|$. In other words, the number of instructions executed by $\mathcal{E}$ between the time that the i-th and the $(i+1)$-th $\#$ are written is linear on the size of y_i. Note that, in particular, an output-linear delay implies that the enumeration phase ends in constant time if there is no output for enumerating.

As the last ingredient, we define how to measure the memory space of a streaming evaluation. Note that after the n-th call a streaming evaluation algorithm with f-update time will necessarily use at most $\mathcal{O}(n \cdot f(|q|))$ bits of space. As a refinement of this bound, we say that this algorithm uses g-space over a query q and stream $\mathcal{S}$ if the number of bits used by it after the n-th call is in $\mathcal{O}(g(|q|, \mathcal{S}[1,n]))$.

Given a streaming enumeration problem, we say that it can be solved with update-time f, output-linear delay, and in g-space if there exists an algorithm such as the one described above. For Δ-enumeration, the notion of streaming evaluation algorithm also applies, even though it could be the case that one can find such an algorithm for full-enumeration but not for Δ-enumeration, and vice versa. Finally, the enumeration problem and solutions provided here are a formal refinement of the algorithmic notions proposed in the literature of streaming evaluation [29], dynamic query evaluation [16, 34], and complex event processing [33, 32].

4 Visibly pushdown transducers and main result

In this section, we present the definition of visibly pushdown transducers [25] (VPT), which are an extension of visibly pushdown automata to produce outputs. We use VPT as our computational model to represent queries with output. This model is general enough to include any query language for nested documents, like XML or JSON, whose expressive power is in MSO. After the setting is formalized, we state the main result of the paper.

A *visibly pushdown transducer* (VPT) is a tuple $\mathcal{T} = (Q, \Sigma, \Gamma, \Omega, \Delta, I, F)$ where Q, Σ, Γ, I, and F are the same as for VPA, Ω is the output alphabet with $\varepsilon \notin \Omega$, and $\Delta \subseteq (Q \times \Sigma^{<} \times (\Omega \cup \{\varepsilon\}) \times Q \times \Gamma) \cup (Q \times \Sigma^{>} \times (\Omega \cup \{\varepsilon\}) \times \Gamma \times Q) \cup (Q \times \Sigma^{|} \times (\Omega \cup \{\varepsilon\}) \times Q)$ is the transition relation. As usual for transducers, a symbol $s \in \Sigma^{<} \cup \Sigma^{>} \cup \Sigma^{|}$ is an input symbol that the machine reads and $o \in \Omega \cup \{\varepsilon\}$ is a symbol that the machine prints in

an output tape. Furthermore, ε represents that no symbol is printed for that transition. A run ρ of $\mathcal{T}$ over a well-nested word $w = s_1 s_2 \cdots s_n \in \Sigma^{\langle *\rangle}$ is a sequence of the form $\rho = (q_1, \sigma_1) \xrightarrow{s_1/o_1} \ldots \xrightarrow{s_n/o_n} (q_{n+1}, \sigma_{n+1})$ where $q_i \in Q$, $\sigma_i \in \Gamma^*$, $q_1 \in I$, $\sigma_1 = \varepsilon$ and for every $i \in [1, n]$ the following holds: (1) if $s_i \in \Sigma^{\langle}$, then $(q_i, s_i, o_i, q_{i+1}, \gamma) \in \Delta$ for some $\gamma \in \Gamma$ and $\sigma_{i+1} = \gamma\sigma_i$, (2) if $s_i \in \Sigma^{\rangle}$, then $(q_i, s_i, o_i, \gamma, q_{i+1}) \in \Delta$ for some $\gamma \in \Gamma$ and $\sigma_i = \gamma\sigma_{i+1}$, and (3) if $s_i \in \Sigma^{|}$, then $(p_i, s_i, o_i, q_{i+1}) \in \Delta$ and $\sigma_i = \sigma_{i+1}$. We say that the run is accepting if $q_{n+1} \in F$. We call a pair (q_i, σ_i) a configuration of ρ. Finally, the output of an accepting run ρ is defined as: $\mathsf{out}(\rho) = \mathsf{out}(o_1, 1) \cdot \ldots \cdot \mathsf{out}(o_n, n)$ where $\mathsf{out}(o, i) = \varepsilon$ when $o = \varepsilon$ and (o, i) otherwise. Note that in $o_1 \cdots o_n$ we use ε as a symbol, and in $\mathsf{out}(\rho)$ we use ε as the empty string. Given a VPT $\mathcal{T}$ and a $w \in \Sigma^{\langle *\rangle}$, we define the set $[\![\mathcal{T}]\!](w)$ of all outputs of $\mathcal{T}$ over w as: $[\![\mathcal{T}]\!](w) = \{\mathsf{out}(\rho) \mid \rho \text{ is an accepting run of } \mathcal{T} \text{ over } w\}$.

Strictly speaking, our definition of VPT is richer than the one studied in [25]. In our definition of VPT each output element is a tuple (o, i) where o is the symbol and i is the output position, where for a standard VPT [25] an output element is just the symbol o. The extension presented here is indeed important for practical applications like in document spanners [26, 6] or in XML query evaluation [12, 46].

A first reasonable question is to understand what is the expressive power of VPT, namely, as a formalism for non-boolean query evaluation over nested words. For the Boolean case, it was shown [4] that VPA describe the same class of queries as MSO over nested words, called $\mathrm{MSO}_{\mathsf{match}}$. Formally, fix a structured alphabet Σ and let $w \in \Sigma^{\langle *\rangle}$ be a word of length n. We encode w as a structure:

$$\big([1, n],\, \leq,\, \{P_a\}_{a \in \Sigma},\, \mathsf{match}\big)$$

where $[1, n]$ is the domain, $\leq$ is the total order over $[1, n]$, $P_a = \{i \mid w[i] = a\}$, and match is a binary relation over $[1, n]$ that corresponds to the matching relation of open and close symbols: $\mathsf{match}(i, j)$ is true iff $w[i]$ is an open symbol and $w[j]$ is its matching close symbol. A $\mathrm{MSO}_{\mathsf{match}}$ formula φ over Σ is given by:

$$\varphi \;:=\; P_a(x) \;\mid\; x \in X \;\mid\; x \leq y \;\mid\; \mathsf{match}(x, y) \;\mid\; \neg\varphi \;\mid\; \varphi \vee \varphi \;\mid\; \exists x.\varphi \;\mid\; \exists X.\varphi$$

where $a \in \Sigma$, x and y are first-order variables and X is a monadic second order (MSO) variable. We write $\varphi(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are the free MSO variables of φ (first-order variables are a special case of MSO variables). Then we write $w \models \varphi(A_1, \ldots, A_n)$ for $A_1, \ldots, A_n \subseteq [1, n]$ when w satisfies φ by replacing each variable X_i with the set A_i. Here, we assume the standard semantics for MSO logic [39].

Given that VPT is an extension of VPA, it should not be a surprise that we can translate these results to VPT. In particular, the result in [4] can be easily extended to link VPT with formulas expressible in $\mathrm{MSO}_{\mathsf{match}}$.

▶ **Proposition 1.** *Let $\varphi(X_1, \ldots, X_m)$ be a $\mathrm{MSO}_{\mathsf{match}}$ formula with m free variables $X_1, \ldots, X_m$. There is a VPT $\mathcal{T}$ for which there is a one-to-one correspondence between the set $[\![\mathcal{T}]\!](w)$ and the set $\{(A_1, \ldots, A_m) \mid w \models \varphi(A_1, \ldots, A_m)\}$ for any word $w \in \Sigma^{\langle *\rangle}$. Moreover, for every VPT $\mathcal{T}$ there is an $\mathrm{MSO}_{\mathsf{match}}$ formula $\varphi(X_1, \ldots, X_m)$ for which the same one-to-one correspondence holds.*

In other words, VPT has the same expressive power as MSO over nested words. Given that fragments of query languages over nested documents (e.g., navigational XPath [47], JSON Navigational Logic [17]) are usually included in MSO, this shows that VPT is an expressive formalism for query evaluation over nested documents.

We say that a VPT $\mathcal{T} = (Q, \Sigma, \Gamma, \Omega, \Delta, I, F)$ is *input/output deterministic* (I/O-deterministic for short) if $|I| = 1$ and Δ is a partial function of the form $\Delta : (Q \times \Sigma^{\langle} \times \Omega \to Q \times \Gamma) \cup (Q \times \Sigma^{\rangle} \times \Omega \times \Gamma \to Q) \cup (Q \times \Sigma^{|} \times \Omega \to Q)$. On the other hand, we say that $\mathcal{T}$ is *input/output unambiguous* (I/O-unambiguous for short) if for every $w \in \Sigma^{\langle *\rangle}$ and every $\mu \in [\![\mathcal{T}]\!](w)$ there is exactly one accepting run ρ of $\mathcal{T}$ over w such that $\mu = \mathsf{out}(\rho)$. Notice that an I/O-deterministic VPT is also I/O-unambiguous and in both models for each output there exists at most one run. The definition of I/O-deterministic is in line with the notion of I/O-deterministic variable automata of [26] and I/O-unambiguous is a generalization of this idea that is enough for the purpose of our enumeration algorithm. One can show that for every VPT $\mathcal{T}$ there exists an equivalent I/O-deterministic VPT and, therefore, an equivalent I/O-unambiguous VPT.

▶ **Lemma 2.** *For every VPT $\mathcal{T}$ there exists an I/O-deterministic VPT $\mathcal{T}'$ of size $\mathcal{O}(2^{|Q|^2|\Gamma|})$ such that $[\![\mathcal{T}]\!](w) = [\![\mathcal{T}']\!](w)$ for every $w \in \Sigma^{\langle *\rangle}$.*

In this paper, we are interested on the following streaming enumeration problem for VPT. Let $\mathcal{C}$ be a class of VPT (e.g. I/O-deterministic VPT).

Problem:	EnumVPT[$\mathcal{C}$]
Input:	a VPT $\mathcal{T} \in \mathcal{C}$ and $w \in \Sigma^{\langle *\rangle}$
Output:	Enumerate $[\![\mathcal{T}]\!](w)$

The main result of the paper is that for the class of I/O-unambiguous VPT, the streaming full-enumeration version of this problem can be solved efficiently.

▶ **Theorem 3.** *The streaming full-enumeration problem of EnumVPT for the class of I/O-unambiguous VPT can be solved with update-time $\mathcal{O}(|Q|^2|\Delta|)$ and output-linear delay. For the general class of VPT, it can be solved with update-time $\mathcal{O}(2^{|Q|^2|\Delta|})$ and output-linear delay.*

The result for the class of all VPT is a consequence of Lemma 2 and the enumeration algorithm for I/O-unambiguous VPT (see Section 5 and 6). For both cases, if the VPT is fixed (i.e., in data complexity), then the update-time of the streaming algorithm is constant.

For the streaming version of EnumVPT, one can have Δ-enumeration with a small loss of efficiency by solving the full-enumeration problem. Specifically, one can show that for any I/O-unambiguous VPT $\mathcal{T}$ there is an I/O-unambiguous VPT $\mathcal{T}'$ of linear size with respect to $|\mathcal{T}|$ such that $[\![\mathcal{T}']\!](w) = [\![\mathcal{T}]\!](w) \setminus \bigcup \{[\![\mathcal{T}]\!](w[1,i]) \mid i < |w|, w[1,i] \in \Sigma^{\langle *\rangle}\}$ for each $w \in \Sigma^{\langle *\rangle}$.

▶ **Theorem 4.** *The streaming Δ-enumeration problem of EnumVPT for the class of I/O-unambiguous VPT can be solved with update-time $\mathcal{O}(|Q|^2|\Delta|)$ and output-linear delay. For the general class of VPT, it can be solved with update-time $\mathcal{O}(2^{|Q|^2|\Delta|})$ and output-linear delay.*

We could have considered a more general definition of VPT to produce outputs for prefix-nested words. This would be desirable for having some sort of *earliest query answering* [29] which is important in practical scenarios. We remark that the algorithm of Theorem 3 can be extended for this case at the cost of making the presentation more complicated. For the sake of presentation, we defer this extension to the full version of this paper.

Space lower bounds of evaluating a VPT. This subsection deals with the space used by the streaming evaluation algorithm of Theorem 3. Indeed, this algorithm could use linear space in the worst case. In the following we explore some lower bounds in the space needed by any algorithm, and show that this bound is tight for a certain type of VPT.

To study the minimum number of bits needed to solve EnumVPT we need to introduce some definitions. Fix a VPT $\mathcal{T}$ and $w \in \mathsf{prefix}(\Sigma^{\langle *\rangle})$. Let $\mathsf{outputweight}(\mathcal{T}, w)$ be the number of positions less than $|w|$ that appear in some output of $[\![\mathcal{T}]\!](w \cdot w')$ for some $w \cdot w' \in \Sigma^{\langle *\rangle}$. Furthermore, for a well-nested word u let $\mathsf{depth}(u)$ be the maximum number of nesting pairs inside u, formally, $\mathsf{depth}(a) = 0$ for $a \in \Sigma^{\mathsf{l}} \cup \{\varepsilon\}$, $\mathsf{depth}(u_1 \cdot u_2) = \max\{\mathsf{depth}(u_1), \mathsf{depth}(u_2)\}$, and $\mathsf{depth}(\langle a \cdot u \cdot b\rangle) = \mathsf{depth}(u) + 1$. For $w \in \mathsf{prefix}(\Sigma^{\langle *\rangle})$, we define $\mathsf{depth}(w) = \min\{\mathsf{depth}(w') \mid w \text{ is a prefix of } w'\}$. We can now state some worst-case space lower bounds for EnumVPT.

▶ **Proposition 5.**

1. *There exists a VPT $\mathcal{T}$ such that every streaming evaluation algorithm for EnumVPT with input $\mathcal{T}$ and $\mathcal{S}$ requires at least $\Omega(\mathsf{depth}(\mathcal{S}[1,n]))$ bits of space.*
2. *There exists a VPT $\mathcal{T}$ such that every streaming evaluation algorithm for EnumVPT with input $\mathcal{T}$ and $\mathcal{S}$ requires at least $\Omega(\mathsf{outputweight}(\mathcal{T}, \mathcal{S}[1,n]))$ bits of space.*

In [12, 13], the authors provide lower bounds on the amount of space needed for evaluating XPath in terms of the nesting and the concurrency (see [12] for a definition). One can show that the output weight of $\mathcal{T}$ and w is always above the concurrency of $\mathcal{T}$ and w. Despite this, one can check that both notions coincide for the space lower bound given in Proposition 5.

The previous results show that, in the worst case, any streaming evaluation algorithm for VPT will require space of at least the depth of the document or the output weight. To show that Theorem 3 is optimal in the worst-case, we need to consider a further assumption of our VPT. We say that a VPT $\mathcal{T}$ is *trimmed* [18] if for every $w \in \mathsf{prefix}(\Sigma^{\langle *\rangle})$ and every (partial) run ρ of $\mathcal{T}$ over w, there exists w' and an accepting run ρ' of $\mathcal{T}$ over $w \cdot w'$ such that ρ is a prefix of ρ'. This notion is the analog of trimmed non-deterministic automata. Similarly to Lemma 2, one can show that for every VPT $\mathcal{T}$ there exists a trimmed I/O-deterministic VPT $\mathcal{T}'$ equivalent to $\mathcal{T}$ (i.e., by extending the construction in [18] to VPT). The next result shows that, if the input to EnumVPT is a trimmed I/O-unambiguous VPT, then the memory footprint is at most the maximum between the depth and output weight of the input.

▶ **Proposition 6.** *The streaming enumeration problem of EnumVPT for the class of trimmed I/O-unambiguous VPT can be solved with update-time $\mathcal{O}(|Q|^2|\Delta|)$, output-linear delay and $\mathcal{O}(\max\{\mathsf{depth}(\mathcal{S}[1,n]), \mathsf{outputweight}(\mathcal{T}, \mathcal{S}[1,n])\} \times |Q|^2|\Delta|)$ space for every stream $\mathcal{S}$.*

Unfortunately, the algorithm provided in Theorem 3 is not *instance optimal*, in the sense of using the lowest number of bits needed for each specific VPT. Note that an instance optimal algorithm for the streaming enumeration problem of VPT will imply a solution to the weak evaluation problem, stated by Segoufin and Vianu [45]. This is an open problem in the area (see [14] for some recent results), so we leave this for future work.

5 Enumerable compact sets: a data structure for output-linear delay

This section presents a data structure, called Enumerable Compact Set (ECS), which is the cornerstone of our enumeration algorithm for VPT. This data structure is strongly inspired by the work in [5, 6]. Indeed, ECS can be considered a refinement of the d-DNNF circuits used in [5] or of the set circuits used in [6]. Several papers [43, 5, 7, 48] have considered circuits-like structures for encoding outputs and enumerate them with constant delay. The novelty of ECS is twofold. First, we use ECS for solving a streaming evaluation problem. Although people have studied streaming query evaluation with enumeration before [34, 16], this is the first work that uses a circuit-like data structure in an online setting. Second and

more important, there is a difference in performance if we compare ECS to the previous approaches. In offline evaluation, constant delay algorithms usually create an initial circuit from the input, making several passes over the structure, building indices, and then running the enumeration process. Given time restrictions for the online evaluation, we cannot create a circuit and do this linear-time preprocessing before enumerating. On the contrary, we must extend the circuit-like data structure for each data item in constant time and then be ready to start the enumeration. This requirement justifies the need for a new data structure for representing and enumerating outputs. Therefore, ECS differs from previous proposals because each operation must take constant time, and we can run the enumeration process with output-linear delay, at any time and without any further preprocessing. In the following, we present ECS step-by-step to use it later in the next section.

Let Σ be a (possibly infinite) alphabet. We define an *Enumerable Compact Set* (ECS) as a tuple $\mathcal{D} = (\Sigma, V, I, \ell, r, \lambda)$ such that V and $I \subseteq V$ are finite sets of nodes, $\ell \colon I \to V$ and $r \colon I \to V$ are the *left* and *right* functions, and $\lambda \colon V \to \Sigma \cup \{\cup, \odot\}$ is a label function such that $\lambda(v) \in \{\cup, \odot\}$ if, and only if, $v \in I$. Further, we assume that the directed graph $(V, \{(v, \ell(v)), (v, r(v)) \mid v \in V\})$ is acyclic. We call the nodes in I *inner nodes* and the nodes in $V \setminus I$ *leaves*. Furthermore, for $v \in I$ we say that v is a *product node* if $\lambda(v) = \odot$, and a *union node* if $\lambda(v) = \cup$. We define the size of $\mathcal{D}$ as $|\mathcal{D}| = |V|$. For each node v in $\mathcal{D}$, we associate a set of words $\mathcal{L}_{\mathcal{D}}(v)$ recursively as follows: (1) $\mathcal{L}_{\mathcal{D}}(v) = \{a\}$ whenever $\lambda(v) = a \in \Sigma$, (2) $\mathcal{L}_{\mathcal{D}}(v) = \mathcal{L}_{\mathcal{D}}(\ell(v)) \cup \mathcal{L}_{\mathcal{D}}(r(v))$ whenever $\lambda(v) = \cup$, and (3) $\mathcal{L}_{\mathcal{D}}(v) = \mathcal{L}_{\mathcal{D}}(\ell(v)) \cdot \mathcal{L}_{\mathcal{D}}(r(v))$ whenever $\lambda(v) = \odot$, where $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$.

The size $|\mathcal{L}_{\mathcal{D}}(v)|$ can be exponential with respect to $|\mathcal{D}|$. For this reason, we say that $\mathcal{D}$ is a *compact* representation of $\mathcal{L}_{\mathcal{D}}(v)$ for any $v \in V$. Although $\mathcal{L}_{\mathcal{D}}(v)$ is very large, the goal is to enumerate all of its elements efficiently. Specifically, we consider the following problem:

Problem:	ENUM-ECS
Input:	An ECS $\mathcal{D} = (\Sigma, V, I, \ell, r, \lambda)$ and $v \in V$.
Output:	Enumerate the set $\mathcal{L}_{\mathcal{D}}(v)$ without repetitions.

Plus, we want to solve ENUM-ECS with output-linear delay. To reach this goal we need to impose two additional restrictions on $\mathcal{D}$. The first restriction is to guarantee that $\mathcal{D}$ is not ambiguous, namely, for each $w \in \mathcal{L}_{\mathcal{D}}(v)$ there is at most one way to retrieve w from $\mathcal{D}$. Formally, we say that $\mathcal{D}$ is *unambiguous* if $\mathcal{D}$ satisfies the following two properties: (1) for every union node v it holds that $\mathcal{L}_{\mathcal{D}}(\ell(v))$ and $\mathcal{L}_{\mathcal{D}}(r(v))$ are disjoint, and (2) for every product node v and for every $w \in \mathcal{L}_{\mathcal{D}}(v)$, there exists a unique way to decompose $w = w_1 \cdot w_2$ such that $w_1 \in \mathcal{L}_{\mathcal{D}}(\ell(v))$ and $w_2 \in \mathcal{L}_{\mathcal{D}}(r(v))$. Thus, if $\mathcal{D}$ is unambiguous, there will be no duplicates if we enumerate $\mathcal{L}_{\mathcal{D}}(v)$ directly, given that there is no way of producing the same element in two different ways.

The second restriction is to guarantee that, for each node v, there exists an output or, more specifically, a symbol of an output *close* to v, in the sense that it can be reached in a bounded number of steps. This is not always the case for an ECS. For example, consider a balanced tree of union nodes where all the outputs are at the leaves. One has to traverse a logarithmic number of nodes from the root to reach the first output. Note that product nodes do not pose this problem since the number of nodes that have to be traversed to produce a certain output is proportional to its length. For this reason, we define the notion of *k-bounded* ECS. Given an ECS $\mathcal{D}$, define the (left) output-depth of a node $v \in V$, denoted by $\mathsf{odepth}_{\mathcal{D}}(v)$, recursively as follows: $\mathsf{odepth}_{\mathcal{D}}(v) = 0$ whenever $\lambda(v) \in \Sigma$ or $\lambda(v) = \odot$, and $\mathsf{odepth}_{\mathcal{D}}(v) = \mathsf{odepth}_{\mathcal{D}}(\ell(v)) + 1$ whenever $\lambda(v) = \cup$. Then, for a fixed $k \in \mathbb{N}$ we say that $\mathcal{D}$ is k-bounded if $\mathsf{odepth}_{\mathcal{D}}(v) \leq k$ for all $v \in V$.

Given the definition of output-depth, we say that v is an output node of $\mathcal{D}$ if v is a leaf or a product node. Note that if $\mathcal{D}$ only has output nodes, then it is 0-bounded, and one can easily check that $\mathcal{L}_{\mathcal{D}}(v)$ can be enumerated with output-linear delay. Indeed, for a fixed k the same happens with every unambiguous and k-bounded ECS.

▶ **Proposition 7.** *Fix $k \in \mathbb{N}$. Let $\mathcal{D} = (\Sigma, V, I, \ell, r, \lambda)$ be an unambiguous and k-bounded ECS. Then the set $\mathcal{L}_{\mathcal{D}}(v)$ can be enumerated with output-linear delay for any $v \in V$.*

The enumeration algorithm above does not require any preprocessing over $\mathcal{D}$ and the main idea is to perform some sort of DFS traversal over the nodes. By this proposition, from now we assume that all ECS are unambiguous and k-bounded for some fixed k.

The next step is to provide a set of operations that allow extending an ECS $\mathcal{D}$ while maintaining k-boundedness. Furthermore, we require these operations to be fully-persistent: a data structure is called *fully-persistent* if every version can be both accessed and modified [22]. In other words, the previous version of the data structure is always available after each operation. To satisfy the last requirement, the strategy will consist in extending $\mathcal{D}$ to $\mathcal{D}'$ for each operation, by always adding new nodes and maintaining the previous nodes untouched. Then $\mathcal{L}_{\mathcal{D}'}(v) = \mathcal{L}_{\mathcal{D}}(v)$ for each node $v \in V$, and so, the structure is fully-persistent.

More precisely, fix an ECS $\mathcal{D} = (\Sigma, V, I, \ell, r, \lambda)$. In the following, we say that $\mathcal{D}' = (\Sigma, V', I', \ell', r', \lambda')$ is an extension of $\mathcal{D}$ if, and only if, $\mathsf{obj} \subseteq \mathsf{obj}'$ for every $\mathsf{obj} \in \{V, I, \ell, r, \lambda\}$. Further, we write $\mathsf{op}(I) \to O$ to define the signature of an operation op where I is the input and O is the output. Then for any $a \in \Sigma$ and $v_1, \ldots, v_4 \in V$, we define the operations:

$$\mathsf{add}(\mathcal{D}, a) \to (\mathcal{D}', v') \qquad \mathsf{prod}(\mathcal{D}, v_1, v_2) \to (\mathcal{D}', v') \qquad \mathsf{union}(\mathcal{D}, v_3, v_4) \to (\mathcal{D}', v')$$

such that $\mathcal{D}'$ is an extension of $\mathcal{D}$ and $v' \in V' \setminus V$ is a fresh node such that $\mathcal{L}_{\mathcal{D}'}(v') = \{a\}$, $\mathcal{L}_{\mathcal{D}'}(v') = \mathcal{L}_{\mathcal{D}}(v_1) \cdot \mathcal{L}_{\mathcal{D}}(v_2)$, and $\mathcal{L}_{\mathcal{D}'}(v') = \mathcal{L}_{\mathcal{D}}(v_3) \cup \mathcal{L}_{\mathcal{D}}(v_4)$, respectively. We assume that the union and prod respect properties (1) and (2) of an unambiguous ECS, that is, $\mathcal{L}_{\mathcal{D}}(v_1)$ and $\mathcal{L}_{\mathcal{D}}(v_2)$ are disjoint and, for every $w \in \mathcal{L}_{\mathcal{D}}(v_3) \cdot \mathcal{L}_{\mathcal{D}}(v_4)$, there exists a unique way to decompose $w = w_1 \cdot w_2$ such that $w_1 \in \mathcal{L}_{\mathcal{D}}(v_3)$ and $w_2 \in \mathcal{L}_{\mathcal{D}}(v_4)$.

Next, we show how to implement each operation. In fact, the case of add and prod are straightforward. For $\mathsf{add}(\mathcal{D}, a) \to (\mathcal{D}', v')$ define $V' := V \cup \{v'\}$, $I' := I$, and $\lambda'(v') = a$. One can easily check that $\mathcal{L}_{\mathcal{D}'}(v') = \{a\}$ as expected. For $\mathsf{prod}(\mathcal{D}, v_1, v_2) \to (\mathcal{D}', v')$ we proceed in a similar way: define $V' := V \cup \{v'\}$, $I' := I \cup \{v\}$, $\ell'(v') := v_1$, $r'(v') = v_2$, and $\lambda'(v') = \odot$. Then $\mathcal{L}_{\mathcal{D}'}(v') = \mathcal{L}_{\mathcal{D}}(v_1) \cdot \mathcal{L}_{\mathcal{D}}(v_2)$. Furthermore, one can check that each operation takes constant time, $\mathcal{D}'$ is a valid ECS (i.e. unambiguous and k-bounded), and the operations are fully-persistent (i.e. the previous version $\mathcal{D}$ is available).

To define the union, we need to be a bit more careful to guarantee output-linear delay, specifically, the k-bounded property. For a node $v \in V$, we say that v is *safe* if (1) $\mathsf{odepth}_{\mathcal{D}}(v) \leq 1$, and (2) if $\mathsf{odepth}_{\mathcal{D}}(v) = 1$, then $\mathsf{odepth}_{\mathcal{D}}(r(v)) \leq 1$. In other words, v is safe if v is an output node, or its left child is an output node, and the right child is either an output node or has output depth 1. Note that a leaf or a product node are safe nodes by definition and, thus, the add and prod operations always produce safe nodes. The trick then is to show that, if v_3 and v_4 are safe nodes, then we can implement $\mathsf{union}(\mathcal{D}, v_3, v_4) \to (\mathcal{D}', v')$ and produce a safe node v'. For this define $(\mathcal{D}', v')$ as follows:

- If v_3 or v_4 are output nodes then $V' := V \cup \{v'\}$, $I' := I \cup \{v'\}$, and $\lambda(v') := \cup$. Moreover, if v_3 is the output node, then $\ell'(v') := v_3$ and $r'(v') := v_4$. Otherwise, we connect $\ell'(v') := v_4$ and $r'(v') := v_3$.
- If v_3 and v_4 are not output nodes (i.e. both are union nodes), then $V' := V \cup \{v', u_1, u_2\}$, $I' := I \cup \{v', u_1, u_2\}$, $\ell'(v') := \ell(v_3)$, $r'(v') := u_1$, and $\lambda'(v') := \cup$; $\ell'(u_1) := \ell(v_4)$, $r'(u_1) := u_2$, and $\lambda'(u_1) := \cup$; $\ell'(u_2) := r(v_3)$, $r'(u_2) := r(v_4)$, and $\lambda'(u_2) := \cup$.

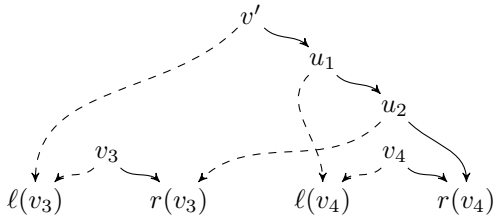


Figure 1 Gadget for $\mathsf{union}(\mathcal{D}, v_3, v_4)$. Nodes v', u_1, u_2, v_3 and v_4 are labeled as $\cup$. Dashed and solid lines denote the mappings in ℓ' and r' respectively.

This gadget is depicted in Figure 1 (note that a similar trick is used in [5] for computing an index over a circuit). This construction has several properties. First, one can easily check that $\mathcal{L}_{\mathcal{D}'}(v') = \mathcal{L}_{\mathcal{D}}(v_1) \cup \mathcal{L}_{\mathcal{D}}(v_2)$ and so the semantics is well-defined. Second, union can be computed in constant time in $|\mathcal{D}|$ given that we only need to add three fresh nodes, and the operation is fully-persistent given that we connect them without modifying $\mathcal{D}$. Furthermore, the produced node v' is safe in $\mathcal{D}'$, although nodes u_1 and u_2 are not necessarily safe. Finally, $\mathcal{D}'$ is 2-bounded whenever $\mathcal{D}$ is 2-bounded. This is straightforward to see for first case when v_3 or v_4 are output nodes. For the second case (i.e., Figure 1), we have to notice that v_3 and v_4 are safe, therefore $\ell(v_3)$ and $\ell(v_4)$ are output nodes, and then $\mathsf{odepth}_{\mathcal{D}'}(v') = \mathsf{odepth}_{\mathcal{D}'}(u_1) = 1$. Further, given that v_3 is safe, we know that $\mathsf{odepth}_{\mathcal{D}}(r(v_3)) \leq 1$, so $\mathsf{odepth}_{\mathcal{D}'}(u_2) \leq 2$. Given that the output depths of all fresh nodes in $\mathcal{D}'$ are bounded by 2 and $\mathcal{D}$ is 2-bounded, then we conclude that $\mathcal{D}'$ is 2-bounded as well.

By the previous discussion, if we start with an ECS $\mathcal{D}$ which is 2-bounded (or empty) and we apply the add, prod and union operators between safe nodes (which also produce safe nodes), then the result is 2-bounded as well. Finally, by Proposition 7, the result can be enumerated with output-linear delay.

▶ **Theorem 8.** *The operations* add, prod, *and* union *require constant time and are fully-persistent. Furthermore, if we start from an empty ECS $\mathcal{D}$ and apply* add, prod, *and* union *over safe nodes, the partial results $(\mathcal{D}', v')$ satisfy that v' is always a safe node and the set $\mathcal{L}_{\mathcal{D}'}(v)$ can be enumerated with output-linear delay for every node v.*

It is important to remark that restricting these operations only over safe nodes is a mild condition. Given that we will usually start from an empty ECS and apply these operations over previously returned nodes, the whole algorithm will always use safe nodes during its computation, satisfying the conditions of Theorem 8.

For technical reasons, our algorithm of the next section needs a slight extension of ECS by allowing leaves that produce the empty string ε. Let $\varepsilon \notin \Sigma$ be a symbol representing the empty string (i.e. $w \cdot \varepsilon = \varepsilon \cdot w = w$). We define an enumerable compact set with ε (called ε-ECS) as a tuple $\mathcal{D} = (\Sigma, V, I, \ell, r, \lambda)$ defined identically to an ECS except that $\lambda : V \to \Sigma \cup \{\cup, \odot, \varepsilon\}$ and $\lambda(v) \in \{\cup, \odot\}$ if, and only, if $v \in I$. Also, if $\lambda(v) = \varepsilon$, then $\mathcal{L}_{\mathcal{D}}(v) = \{\varepsilon\}$. The unambiguity restriction is the same for ε-ECS and one has to slightly extend k-boundedness to consider ε-nodes. However, to support the prod and union operations in constant time and to maintain the k-boundedness invariant, we need to extend the notion of safe nodes (called ε-safe) and the gadgets for prod and union.

▶ **Theorem 9.** *The operations* add, prod, *and* union *over ε-ECS take constant time and are fully-persistent. Furthermore, if we start from an empty ε-ECS $\mathcal{D}$ and apply* add, prod, *and* union *over ε-safe nodes, the partial results $(\mathcal{D}', v')$ satisfy that v' is always an ε-safe node and the set $\mathcal{L}_{\mathcal{D}'}(v)$ can be enumerated with output-linear delay for every node v.*

6 Evaluating visibly pushdown transducers with output-linear delay

The goal of this section is to describe an algorithm that takes an I/O-unambiguous VPT $\mathcal{T}$ plus a stream $\mathcal{S}$, and enumerates the set $[\![\mathcal{T}]\!](\mathcal{S}[1,n])$ for an arbitrary $n \geq 0$ with $\mathcal{O}(|Q|^2|\Delta|)$-update-time and output-linear delay. We divide the presentation of the algorithm into two parts. The first part explains the determinization of a VPA, which is instrumental in understanding our update phase. The second part gives the algorithm and proves its correctness. Given that a neutral symbol a can be represented as a pair $\langle a \cdot a \rangle$, in this section we present the algorithm and definitions without neutral letters, that is, the structured alphabet is $\Sigma = (\Sigma^<, \Sigma^>)$. Thus, from now on we use a for denoting any symbol in $\Sigma^< \cup \Sigma^>$.

Determinization of visibly pushdown automata. A significant result in Alur and Madhusudan's paper [4] that introduces VPA was that one can always determinize them. We provide here an alternative proof for this result that requires a somewhat more direct construction. This determinization process is behind our update algorithm and serves to give some crucial notions of how it works. We start by providing the determinization construction, introducing some useful notation, and then giving some intuition.

Given a VPA $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, I, F)$, we define the following deterministic VPA $\mathcal{A}^{\text{det}} = (Q^{\text{det}}, q_0^{\text{det}}, \Gamma^{\text{det}}, \delta^{\text{det}}, F^{\text{det}})$ with state set $Q^{\text{det}} = 2^{Q \times Q}$ and stack symbol set $\Gamma^{\text{det}} = 2^{Q \times \Gamma \times Q}$. The initial state is $q_0^{\text{det}} = \{(q,q) \mid q \in I\}$ and the set of final states is $F^{\text{det}} = \{S \in Q^{\text{det}} \mid S \cap (I \times F) \neq \emptyset\}$. Finally, we define the transition function δ^{det} such that if $\langle a \in \Sigma^<$, then $\delta^{\text{det}}(S, \langle a) = (S', T')$ where $S' = \{(q,q) \mid \exists p, p', \gamma.\ (p,p') \in S \wedge (p', \langle a, q, \gamma) \in \Delta\}$ and $T' = \{(p, \gamma, q) \mid \exists p'.\ (p,p') \in S \wedge (p', \langle a, q, \gamma) \in \Delta\}$; if $a\rangle \in \Sigma^>$, then $\delta^{\text{det}}(S, T, a\rangle) = S'$ where $S' = \{(p,q) \mid \exists p', q', \gamma.\ (p, \gamma, p') \in T \wedge (p', q') \in S \wedge (q', a\rangle, \gamma, q) \in \Delta\}$.

To understand the purpose of this construction, first we need to introduce some notation. Fix a well-nested word $w = a_1 a_2 \cdots a_n$. A span s of w is a pair $[i,j\rangle$ of natural numbers i and j with $1 \leq i \leq j \leq n+1$. We denote by $w[i,j\rangle$ the subword $a_i \cdots a_{j-1}$ of w and, when $i = j$, we assume that $w[i,j\rangle = \varepsilon$. Intuitively, spans are indexing w with intermediate positions, like $_1a_1{}_2a_2{}_3 \ldots {}_n a_n{}_{n+1}$, where i is between symbols a_{i-1} and a_i. Then $[i,j\rangle$ represents an interval $\{i, \ldots, j\}$ that captures the subword $a_i \ldots a_{j-1}$.

Now, we say that a span $[i,j\rangle$ of w is well-nested if $w[i,j\rangle$ is well-nested. Note that ε is well-nested, so $[i,i\rangle$ is a well-nested span for every i. For a position $k \in [1, n+1]$, we define the *current-level span* of k, $\mathsf{currlevel}(k)$, as the well-nested span $[j,k\rangle$ such that $j = \min\{j' \mid [j',k\rangle \text{ is well-nested}\}$. Note that $[k,k\rangle$ is always well-nested and thus $\mathsf{currlevel}(k)$ is well defined. We also identify the *lower-level span* of k, $\mathsf{lowerlevel}(k)$, defined as $\mathsf{lowerlevel}(k) = \mathsf{currlevel}(j-1) = [i, j-1\rangle$ whenever $\mathsf{currlevel}(k) = [j,k\rangle$ and $j > 1$. In contrast to $\mathsf{currlevel}(k)$, $\mathsf{lowerlevel}(k)$ is not always well-defined given that it is "one level below" than $\mathsf{currlevel}(k)$ and this may not exist. More concretely, for $\mathsf{currlevel}(k) = [j,k\rangle$ and $\mathsf{lowerlevel}(k) = [i, j-1\rangle$, these spans will look as follows:

$$_1a_1{}_2a_2{}_3 \ldots {<}a_{i-1}{}_i \overbrace{a_i \ldots a_{j-2}}^{\mathsf{lowerlevel}(k)}{}_{j-1} {<}a_{j-1}{}_j \overbrace{a_j \ldots a_{k-1}}^{\mathsf{currlevel}(k)} \downarrow_k a_k \ldots {}_n a_n{}_{n+1}$$

As an example, consider the word $_1($ $_2($ $_3)$ $_4($ $_5($ $_6)$ $_7)$ $_8)$ $_9$. The only well-nested spans besides the ones of the form $[i,i\rangle$ are $[1,9\rangle$, $[2,4\rangle$, $[2,8\rangle$, $[4,8\rangle$ and $[5,7\rangle$, therefore $\mathsf{currlevel}(8) = [2,8\rangle$, and $\mathsf{lowerlevel}(7) = [2,4\rangle$.

We are ready to explain the purpose of the determinization above. Let $w = a_1 a_2 \cdots a_n$ be a well-nested word and $\rho^{\text{det}} = (S_1, \tau_1) \xrightarrow{a_1} \ldots \xrightarrow{a_{k-1}} (S_k, \tau_k)$ be the (partial) run of $\mathcal{A}^{\text{det}}$ until some k. Furthermore, assume $\tau_k = T_k \cdot \tau$ for some $T_k \in \Gamma^{\text{det}}$ and $\tau \in (\Gamma^{\text{det}})^*$. The connection between ρ^{det} and the runs of $\mathcal{A}$ over $a_1 \ldots a_{k-1}$ is given by the following invariants:

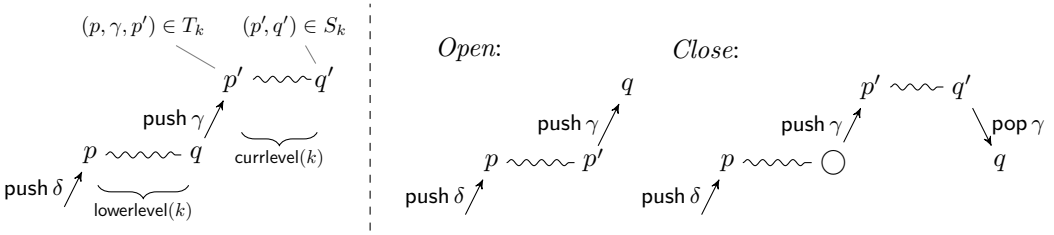


Figure 2 Left: An example run of some VPA $\mathcal{A}$ at step k. Right: Illustration of two nondeterministic runs for some VPA $\mathcal{A}$, as considered in the determinization process.

(a) $(p, q) \in S_k$ if, and only if, there exists a run $(q_1, \sigma_1) \xrightarrow{a_1} \dots \xrightarrow{a_{k-1}} (q_k, \sigma_k)$ of $\mathcal{A}$ over $a_1 \dots a_{k-1}$ such that $q_j = p$, $q_k = q$, and $\mathsf{currlevel}(k) = [j, k\rangle$.

(b) $(p, \gamma, q) \in T_k$ if, and only if, there exists a run $(q_1, \sigma_1) \xrightarrow{a_1} \dots \xrightarrow{a_{k-1}} (q_k, \sigma_k)$ of $\mathcal{A}$ over $a_1 \dots a_{k-1}$ such that $q_i = p$, $q_j = q$, $\sigma_k = \gamma\sigma$ for some σ, and $\mathsf{lowerlevel}(k) = [i, j-1\rangle$.

On one hand, (a) says that each pair $(p, q) \in S_k$ represents some non-deterministic run of $\mathcal{A}$ over w for which q is the k-th state, and p was visited on the step when the current symbol at the top of the stack was pushed. On the other hand, (b) says that $(p, \gamma, q) \in T_k$ represents some run of $\mathcal{A}$ over w for which γ is at the top of the stack, q was visited on the step when γ was pushed, and p was visited on the step when the symbol below γ was pushed (see Figure 2 (left)). More importantly, these conditions are exhaustive, that is, every run of $\mathcal{A}$ over $a_1 \dots a_{k-1}$ is represented by ρ^{det}.

By these two invariants, the correctness of $\mathcal{A}^{\text{det}}$ easily follows and the reader can get some intuition behind $\delta^{\text{det}}(S, \langle a)$ and $\delta^{\text{det}}(S, T, a\rangle)$ (see Figure 2 (right) for a graphical description). Indeed, the most important consequence of these two invariants is that a tuple $(q_j, q_k) \in S_k$ represents the interval of some run over $w[j, k\rangle$ with $\mathsf{currlevel}(k) = [j, k\rangle$ and the tuple $(q_i, \gamma, q_j) \in T_k$ represents the interval of some run over $w[i, j-1\rangle$ with $\mathsf{lowerlevel}(k) = [i, j-1\rangle$, i.e., the level below. In other words, the configuration (S_k, τ_k) of $\mathcal{A}^{\text{det}}$ forms a succinct representation of all the non-deterministic runs of $\mathcal{A}$. This is the starting point of our update algorithm, that we discuss next.

The streaming evaluation algorithm. In Algorithm 1 we present the update phase for solving the streaming version of EnumVPT. The main procedure is UpdatePhase, that receives an I/O-unambiguous VPT $\mathcal{T} = (Q, \Sigma, \Gamma, \Omega, \Delta, I, F)$ and a stream $\mathcal{S}$, reads the next k-th symbol and computes the set of outputs $[\![\mathcal{T}]\!](\mathcal{S}[1, k])$. More specifically, it constructs an ε-ECS $\mathcal{D}$ and a vertex v_{out} such that $\mathcal{L}_{\mathcal{D}}(v_{\text{out}}) = [\![\mathcal{T}]\!](\mathcal{S}[1, k])$ if $\mathcal{S}[1, k]$ is well-nested and $\emptyset$ otherwise. After the UpdatePhase procedure is done, we can enumerate $\mathcal{L}_{\mathcal{D}}(v_{\text{out}})$ with output-linear delay by calling the enumeration phase, that is, by applying Theorem 9.

Towards this goal, in Algorithm 1 we make use of the following data structures: First of all, we use an ε-ECS $\mathcal{D} = (\Sigma, V, I, \ell, r, \lambda)$, nodes $v \in V$, and the functions add, union, and prod over $\mathcal{D}$ and v (see Section 5). For the sake of simplification, we overload the notation of these operators slightly so that if $v = \emptyset$, then $\mathsf{union}(\mathcal{D}, v, v') = \mathsf{union}(\mathcal{D}, v', v) = (\mathcal{D}, v')$. We use a hash table S which indexes nodes v in $\mathcal{D}$ by pairs of states $(p, q) \in Q \times Q$. We denote the elements of S as "$(p, q) : v$" where (p, q) is the index and v is the content. Furthermore, we write $S_{p,q}$ to access the node v. We also use a stack T that stores hash tables: each element is a hash table which indexes vertices v in $\mathcal{D}$ by triples $(p, \gamma, q) \in Q \times \Gamma \times Q$. We assume that T has the standard stack methods push and pop where if $T = t_k \cdots t_1$, then $\mathsf{push}(T, t) = t\, t_k \cdots t_1$ and $\mathsf{pop}(T) = t_{k-1} \cdots t_1$. We write $\emptyset$ for denoting the empty stack or

for checking if T is empty. Similarly to S, we use the notation $T_{p,\gamma,q}$ to access the nodes in the topmost hash-table in T (i.e. T is a stack of hash tables). We assume that accessing a non-assigned index in these hash tables returns the empty set. All variables (e.g., S, and T) are defined globally in Algorithm 1 and they can be accessed by any of the subprocedures. given that we use the RAM model (see Section 2), each operation over hash tables or stacks takes constant time.

Algorithm 1 builds the ε-ECS $\mathcal{D}$ incrementally, reading $\mathcal{S}$ one letter at a time by calling $\texttt{yield}[\mathcal{S}]$ and keeping a counter k for the position of the current letter. For every $k \in [1, n+1]$, UpdatePhase builds the k-th iteration of table S and stack T, which we note as S^k and T^k respectively. Before UpdatePhase is called for the first time, it runs Intialize (lines 1-4) to set the initial values of k, $\mathcal{D}$, S, and T. We consider the initial S and T as the 1-st iteration, defined as $S^1 = \{(q, q) : v_\varepsilon \mid q \in I\}$ and $T^1 = \emptyset$ (i.e. the empty stack) where v_ε is a node in $\mathcal{D}$ such that $\mathcal{L}_\mathcal{D}(v_\varepsilon) = \{\varepsilon\}$ (lines 3-4). In the k-th iteration, depending on whether the current letter is an open symbol or a close symbol, the OpenStep or CloseStep procedures are called, updating S^{k-1} and T^{k-1} to S^k and T^k, respectively. More specifically, UpdatePhase adds nodes to $\mathcal{D}$ such that the nodes in S^k represent the runs over $w[j, k\rangle$ where $\mathsf{currlevel}(k) = [j, k\rangle$, and the nodes in the topmost table in T^k represent the runs over $w[i, j-1\rangle$ where $\mathsf{lowerlevel}(k) = [i, j-1\rangle$. Moreover, for a given pair (p, q), the node $S^k_{p,q}$ represents all runs over $w[j, k\rangle$ with $\mathsf{currlevel}(k) = [j, k\rangle$ that start on p and end on q. For a given triple (p, γ, q) the node $T^k_{p,\gamma,q}$ represents all runs over $w[i, j-1\rangle$ with $\mathsf{lowerlevel}(k) = [i, j-1\rangle$ that start on p, and end on q right after pushing γ onto the stack. Here, the intuition gained in the determinization of VPA is crucial. Indeed, table S^k and stack T^k are the mirror of the configuration (S_k, τ_k) of $\mathcal{A}^{\text{det}}$ (recall invariants (a) and (b)).

Before formalizing these notions, we will describe in more detail what the procedures OpenStep and CloseStep exactly do. Recall that the operation $\mathsf{add}(\mathcal{D}, a)$ simply creates a node in $\mathcal{D}$ labeled as a; the operation $\mathsf{prod}(\mathcal{D}, v_1, v_2)$ returns a pair $(\mathcal{D}', v')$ such that $\mathcal{L}_{\mathcal{D}'}(v') = \mathcal{L}_\mathcal{D}(v_1) \cdot \mathcal{L}_\mathcal{D}(v_2)$; and the operation $\mathsf{union}(\mathcal{D}, v_3, v_4)$ returns a pair $(\mathcal{D}', v')$ such that $\mathcal{L}_{\mathcal{D}'}(v') = \mathcal{L}_\mathcal{D}(v_3) \cup \mathcal{L}_\mathcal{D}(v_4)$. To improve the presentation of the algorithm, we include a simple procedure called IfProd (lines 19-25). Basically, this procedure receives a node v, an output symbol o, and a position k, and computes $(\mathcal{D}', v')$ such that $\mathcal{L}_{\mathcal{D}'}(v') = \mathcal{L}_\mathcal{D}(v) \cdot \{(o, k)\}$ if $o \neq \varepsilon$, and $\mathcal{L}_{\mathcal{D}'}(v') = \mathcal{L}_\mathcal{D}(v)$ otherwise.

In OpenStep, S^k is created (i.e. S'), and an empty table is pushed onto T^{k-1} to form T^k (line 27). Then, all nodes in S^{k-1} (i.e. S) are checked to see if the runs they represent can be extended with a transition in Δ (lines 28-29). If this is the case (lines 30 onwards), a node v_ε with the ε-output is added in S^k to start a new level (lines 30-32). Then, if the transition had a non-empty output, the node $S^k_{p,p'}$ is connected with a new label node to form the node v (lines 33-34). This node is stored in $T^k_{p,\gamma,q}$, or united with the node that was already present there (lines 35-36).

In CloseStep, S^k is initialized as empty (line 41). Then, the procedure looks for all of the valid ways to join a node in T^{k-1}, a node in S^{k-1}, and a transition in Δ to form a new node in S^k. More precisely, it looks for quadruples (p, γ, p', q') for which $T^{k-1}_{p,\gamma,p'}$ and $S^{k-1}_{p',q'}$ are defined, and there is a close transition that starts on q' that reads γ (lines 42-43). These nodes are joined and connected with a new label node if it corresponds (lines 44-45), and stored in $S^k_{p,q}$ or united with the node that was already present there (lines 46-47). Finally, the top of the stack T is popped after all tuples (p, γ, p', q') are checked (line 48).

As it was already mentioned, in each step the construction of $\mathcal{D}$ follows the ideas of the determinization of a visibly pushdown automata. As such, Figure 2 also aids to illustrate how the table S^k and the top of the stack T^k are constructed.

Algorithm 1 The update phase of the streaming evaluation algorithm for EnumVPT given an I/O-unambiguous VPT $\mathcal{T} = (Q, \Sigma, \Gamma, \Omega, \Delta, I, F)$ and a stream $\mathcal{S}$.

```
1:  procedure Initialize(T, S)
2:      k ← 1, D ← ∅
3:      (D, v_ε) ← add(D, ε)
4:      S ← {(q, q) : v_ε | q ∈ I}, T ← ∅
5:
6:  procedure UpdatePhase(T, S)
7:      a ← yield[S]
8:      if a ∈ Σ^< then
9:          D ← OpenStep(D, a, k)
10:     else if a ∈ Σ^> then
11:         D ← CloseStep(D, a, k)
12:     k ← k + 1
13:     v_out ← ∅
14:     if T = ∅ then
15:         for each p ∈ I, q ∈ F s.t. S_{p,q} ≠ ∅ do
16:             (D, v_out) ← union(D, v_out, S_{p,q})
17:     EnumerationPhase(D, v_out)
18:
19: procedure IfProd(D, v, o, k)
20:     if o ≠ ε then
21:         (D', v') ← add(D, (o, k))
22:         (D', v') ← prod(D', v, v')
23:     else
24:         (D', v') ← (D, v)
25:     return (D', v')
26: procedure OpenStep(D, <a, k)
27:     S' ← ∅, T ← push(T, ∅)
28:     for p ∈ Q and (p', <a, o, q, γ) ∈ Δ do
29:         if S_{p,p'} ≠ ∅ then
30:             if S'_{q,q} = ∅ then
31:                 (D, v_ε) ← add(D, ε)
32:                 S'_{q,q} ← v_ε
33:             v ← S_{p,p'}
34:             (D, v) ← IfProd(D, v, o, k)
35:             (D, v) ← union(D, v, T_{p,γ,q})
36:             T_{p,γ,q} ← v
37:     S ← S'
38:     return D
39:
40: procedure CloseStep(D, a>, k)
41:     S' ← ∅
42:     for p, p' ∈ Q and (q', a>, o, γ, q) ∈ Δ do
43:         if S_{p',q'} ≠ ∅ and T_{p,γ,p'} ≠ ∅ then
44:             (D, v) ← prod(D, T_{p,γ,p'}, S_{p',q'})
45:             (D, v) ←IfProd(D, v, o, k)
46:             (D, v) ← union(D, v, S'_{p,q})
47:             S'_{p,q} ← v
48:     T ← pop(T)
49:     S ← S'
50:     return D
```

The way how the table S^k and the stack T^k are constructed is formalized in the following result. Recall that a run of $\mathcal{T}$ over a well-nested word $w = a_1 \cdots a_n$ is a sequence of the form $\rho = (q_1, \sigma_1) \xrightarrow{a_1/o_1} \ldots \xrightarrow{a_n/o_n} (q_{n+1}, \sigma_{n+1})$. Given a span $[i, j\rangle$, define a subrun of ρ as a subsequence $\rho[i, j\rangle = (q_i, \sigma_i) \xrightarrow{a_i/o_i} \ldots \xrightarrow{a_{j-1}/o_{j-1}} (q_j, \sigma_j)$. We also extend the function out to receive a subrun $\rho[i, j\rangle$ in the following way: $\mathsf{out}(\rho[i, j\rangle) = \mathsf{out}(o_i, i) \cdot \ldots \cdot \mathsf{out}(o_{j-1}, j-1)$. Finally, define $\mathrm{Runs}(\mathcal{T}, w)$ as the set of all runs of $\mathcal{T}$ over w.

▶ **Lemma 10.** *Let $\mathcal{T}$ be a VPT and $w = a_1 \cdots a_n$ be a well-nested word. While running the procedure* UpdatePhase *of Algorithm 1, for every $k \in [1, n+1]$, every pair of states p, q and stack symbol γ the following hold:*

1. *$\mathcal{L}_{\mathcal{D}}(S^k_{p,q})$ has exactly all sequences $\mathsf{out}(\rho[j, k\rangle)$ such that $\rho \in \mathrm{Runs}(\mathcal{T}, w[1, k\rangle)$, $\mathsf{currlevel}(k) = [j, k\rangle$, and $\rho[j, k\rangle$ starts on p and ends on q.*
2. *If $\mathsf{lowerlevel}(k)$ is defined, then $\mathcal{L}_{\mathcal{D}}(T^k_{p,\gamma,q})$ has exactly all sequences $\mathsf{out}(\rho[i, j\rangle)$ such that $\rho \in \mathrm{Runs}(\mathcal{T}, w[1, j\rangle)$, $\mathsf{lowerlevel}(k) = [i, j-1\rangle$, and $\rho[i, j\rangle$ starts on p, ends on q, and the last symbol pushed onto the stack was γ.*

Since w is well nested, then $\mathsf{currlevel}(|w|+1) = [1, |w|+1\rangle$, and so, the lemma implies that the nodes in $S^{|w|+1}$ represent all runs of $\mathcal{T}$ over w. Then, whenever $\mathcal{S}[1, k]$ is well-nested, the stack T is empty (i.e., $T = \emptyset$) and there may be something to enumerate (line 14). By taking the union of all pairs in S^{k+1} that represent accepting runs (as is done in lines 15-16), we can conclude the following result:

▶ **Theorem 11.** *Given a VPT $\mathcal{T}$ and a stream $\mathcal{S}$,* UpdatePhase$(\mathcal{T}, \mathcal{S})$ *fulfils the conditions of a streaming evaluation algorithm and, after reading the k-th symbol, produces a pair $(\mathcal{D}, v_{\text{out}})$ such that $\mathcal{L}_{\mathcal{D}}(v_{\text{out}}) = [\![\mathcal{T}]\!](\mathcal{S}[1, k])$.*

At this point we address the fact that $\mathcal{D}$ needs to be unambiguous in order to enumerate all the outputs from $(\mathcal{D}, v_{\text{out}})$ without repetitions. This is guaranteed, essentially, by the fact that $\mathcal{T}$ is I/O-unambiguous as well. Indeed, the previous result holds even if $\mathcal{T}$ is not I/O-unambiguous. The next result guarantees that the output can be enumerated efficiently.

▶ **Lemma 12.** *Let $\mathcal{T}$ be an I/O-unambiguous VPT. While running* UpdatePhase *procedure of Algorithm 1, the ε-ECS $\mathcal{D}$ is unambiguous at every step.*

The complexity of this algorithm can be easily deduced from the fact that the ε-ECS operations we use take constant time (Theorem 9). For a VPT $\mathcal{T} = (Q, \Sigma, \Gamma, \Omega, \Delta, I, F)$, in each of the calls to OpenStep, lines 29-36 perform a constant number of instructions, and they are visited at most $|Q||\Delta|$ times. In each of the calls to CloseStep, lines 43-47 perform a constant number of instructions, and they are visited at most $|Q|^2|\Delta|$ times. Combined with Theorem 11, Lemma 12, and Theorem 9, this proves our main result (i.e. Theorem 3).

7 Future work

This paper offers several directions for future work. One direction is to find a streaming evaluation algorithm with polynomial update-time for non-deterministic VPT (i.e., in the size of the VPT). In [6], the authors provided a polynomial-time offline algorithm for non-deterministic word transducers (called vset automata). They extended this result to trees in [7]. One could use these techniques in Algorithm 1; however, it is unclear how to extend ECS to deal with ambiguity in a natural way. Regarding space resources, another direction is to find an "instance optimal" streaming evaluation algorithm for VPT. As we mentioned, this problem generalizes the weak evaluation problem stated in [45], given that it also considers the space to represent the output compactly. Finally, it would be interesting to explore practical implementations. Our view is that the data structure and algorithm presentation aid in reaching this goal, and it leaves space for suitable optimizations.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

2 Mehmet Altınel and Michael J Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB*, pages 53–64, 2000.

3 Rajeev Alur, Dana Fisman, Konstantinos Mamouras, Mukund Raghothaman, and Caleb Stanford. Streamable regular transductions. *Theor. Comput. Sci.*, 807:15–41, 2020.

4 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *STOC*, pages 202–211, 2004.

5 Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *ICALP*, pages 111:1–111:15, 2017.

6 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, pages 22:1–22:19, 2019.

7 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.

8 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. In *PODS*, pages 59–73, 2019.

9 Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *SIGMOD*, pages 1–16, 2002.

10 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL*, pages 167–181, 2006.

11 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL*, pages 208–222, 2007.

12 Ziv Bar-Yossef, Marcus Fontoura, and Vanja Josifovski. Buffering in query evaluation over XML streams. In *PODS*, pages 216–227, 2005.

13 Ziv Bar-Yossef, Marcus Fontoura, and Vanja Josifovski. On the memory requirements of XPath evaluation over XML streams. *J. Comput. Syst. Sci.*, 73(3):391–441, 2007.

14 Corentin Barloy, Filip Murlak, and Charles Paperman. Stackless processing of streamed trees. In *PODS*, 2021.

15 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020.

16 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *PODS*, pages 303–318, 2017.

17 Pierre Bourhis, Juan L. Reutter, and Domagoj Vrgoc. JSON: Data model and query languages. *Inf. Syst.*, 89:101478, 2020.

18 Mathieu Caralp, Pierre-Alain Reynier, and Jean-Marc Talbot. Trimming visibly pushdown automata. *Theor. Comput. Sci.*, 578:13–29, 2015.

19 Yi Chen, Susan B. Davidson, and Yifeng Zheng. An efficient XPath query processor for XML streams. In *ICDE*, page 79, 2006.

20 Rada Chirkova and Jun Yang. Materialized views. *Found. Trends Databases*, 4(4):295–405, 2012.

21 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discret. Appl. Math.*, 157(12):2675–2700, 2009.

22 James R Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. In *STOC*, pages 109–121, 1986.

23 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4):21, 2007.

24 Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. Streamability of nested word transductions. *LMCS*, 15(2), 2019.

25 Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Visibly pushdown transducers. *JCSS*, 97:147–181, 2018.

26 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Efficient enumeration algorithms for regular document spanners. *TODS*, 45(1):3:1–3:42, 2020.

27 Olivier Gauwin, Joachim Niehren, and Yves Roos. Streaming tree automata. *Inf. Process. Lett.*, 109(1):13–17, 2008.

28 Olivier Gauwin, Joachim Niehren, and Sophie Tison. Bounded delay and concurrency for earliest query answering. In *LATA*, volume 5457, pages 350–361, 2009.

29 Olivier Gauwin, Joachim Niehren, and Sophie Tison. Earliest query answering for deterministic nested word automata. In *FCT*, volume 5699, pages 121–132, 2009.

30 Gang Gou and Rada Chirkova. Efficient algorithms for evaluating XPath over streams. In *SIGMOD*, pages 269–280. ACM, 2007.

31 Todd J. Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, and Dan Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM Trans. Database Syst.*, 29(4):752–788, 2004.

32 Alejandro Grez and Cristian Riveros. Towards streaming evaluation of queries with correlation in complex event processing. In *ICDT*, pages 14:1–14:17, 2020.

33 Alejandro Grez, Cristian Riveros, and Martín Ugarte. A formal framework for complex event processing. In *ICDT*, pages 5:1–5:18, 2019.

34 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The dynamic Yannakakis algorithm: Compact and efficient query processing under updates. In *SIGMOD*, pages 1259–1274, 2017.

35 Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

36 Vanja Josifovski, Marcus Fontoura, and Attila Barta. Querying XML streams. *VLDB J.*, 14(2):197–210, 2005.

37 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *PODS*, pages 375–392, 2020.

38 Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Visibly pushdown automata for streaming XML. In *WWW*, pages 1053–1062, 2007.

39 Leonid Libkin. *Elements of finite model theory*, volume 41. Springer, 2004.

40 Milos Nikolic and Dan Olteanu. Incremental view maintenance with triple lock factorization benefits. In *SIGMOD*, pages 365–380, 2018.

41 Dan Olteanu. SPEX: Streamed and progressive evaluation of XPath. *IEEE Trans. Knowl. Data Eng.*, 19(7):934–949, 2007.

42 Dan Olteanu, Tim Furche, and François Bry. An efficient single-pass query evaluator for XML data streams. In *SAC*, pages 627–631, 2004.

43 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM TODS*, 40(1):2:1–2:44, 2015.

44 Luc Segoufin. Enumerating with constant delay the answers to a query. In *ICDT*, pages 10–20, 2013.

45 Luc Segoufin and Victor Vianu. Validating streaming XML documents. In *PODS*, pages 53–64, 2002.

46 Mirit Shalem and Ziv Bar-Yossef. The space complexity of processing XML twig queries over indexed documents. In *ICDE*, pages 824–832, 2008.

47 Balder ten Cate and Maarten Marx. Navigational XPath: calculus and algebra. *SIGMOD Record*, 36(2):19–26, 2007.

48 Szymon Torunczyk. Aggregate queries on sparse databases. In *PODS*, pages 427–443, 2020.