19th International Conference on Database Theory

ICDT'16, March 15–18, 2016, Bordeaux, France

Edited by

Wim Martens

Thomas Zeume

LIPIcs – Vol. 48 – ICDT'16 www.dagstuhl.de/lipics

Editors

Wim Martens
University of Bayreuth
Bayreuth, Germany
wim.martens@uni-bayreuth.de

Thomas Zeume
TU Dortmund University
Dortmund, Germany
thomas.zeume@tu-dortmund.edu

ACM Classification 1998
H.2 Database Management, H.2.1 Normal forms, H.2.2 Schema and subschema, H.2.3 Query languages, H.2.4 Query processing, H.2.4 Relational databases, H.2.4 Distributed databases, H.2.5 Heterogeneous Databases, H.3.5 Online Information Services, H.1 Miscellaneous – Privacy, H.4.1 Office Automation: Workflow management, B.4.4 Performance Analysis and Design Aids: Formal models, Verification, F.1.3 Complexity measures and classes, F.4.1 Computational Logic, Model Theory, G.2.2 Graph Theory|Hypergraphs

ISBN 978-3-95977-002-6

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-002-6.

Publication date
March, 2016

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.ICDT.2016.0

ISBN 978-3-95977-002-6 **ISSN 1868-8969** **http://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

http://www.dagstuhl.de/lipics

Contents

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Optimization

Evolving Data

Data Extraction and Analytics

Schemas and Consistency

SPARQL and Graphs

Preface

The 19th International Conference on Database Theory (ICDT 2016) was held in Bordeaux, France, March 15–18, 2016. Originally biennial, the ICDT conference has been held annually and jointly with EDBT ("Extending Database Technology") since 2009.

The proceedings of ICDT 2016 include an overview of a keynote by Floris Geerts (University of Antwerp), an overview of a keynote by Yufei Tao (University of Queensland), a paper by David P. Woodruff (IBM Almaden) based on his invited lecture, a laudation concerning the ICDT 2016 Test of Time Award, and 19 research papers that were selected by the Program Committee from 41 submissions.

Out of the 19 accepted papers, the Program Committee selected the paper *Beyond Well-Designed SPARQL* by Mark Kaminski and Egor V. Kostylev for the ICDT 2016 Best Paper Award. Furthermore, the Program Committee selected the paper *A Framework for Estimating Stream Expression Cardinalities* by Anirban Dasgupta, Kevin Lang, Lee Rhodes, and Justin Thaler for the ICDT 2016 Best Newcomer Award. The Test of Time Award for ICDT 2016 is given to the paper *Conjunctive Query Containment Revisited* by Chandra Chekuri and Anand Rajaraman, which originally appeared in the proceedings of ICDT 1997. Warmest congratulations to the authors of these award winning papers!

I thank all authors who submitted papers to ICDT 2016. I would also like to thank all members of the Program Committee, and the external reviewers, for the enormous amount of work they have done. The Program Committee carried out extensive discussions during the electronic PC meetings, before and after rebuttal. I am very grateful to Foto Afrati, Claire David, and Georg Gottlob for their efforts and expertise in selecting the paper for the Test of Time Award. I thank Andrei Voronkov for his EasyChair system, which made it easy to manage and coordinate the discussion.

I thank the ICDT Council members for their help in selecting the Program Committee and in particular the Council Chair Thomas Schwentick for his continuous advice on a wide variety of matters concerning ICDT. Special thanks also go to Thomas Zeume, the Proceedings Chair of ICDT 2016. I thank last year's PC Chair Marcelo Arenas and last year's Proceedings Chair Martín Ugarte for sharing their knowledge and experience which substantially helped us in producing the proceedings. Finally, I thank many colleagues involved in the organisation of the conference for fruitful collaboration, in particular, Sofian Maabout (EDBT/ICDT 2016 Conference Chair).

Wim Martens
January 2016

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Conference Chair

Sofian Maabout (U. of Bordeaux)

Program Chair

Wim Martens (U. of Bayreuth)

Program Committee

Yael Amsterdamer (Tel Aviv U.)
Pablo Barceló (U. de Chile)
Hubie Chen (U. del Pais Vasco and Ikerbasque)
Sara Cohen (Hebrew U. of Jerusalem)
Graham Cormode (U. of Warwick)
Wenfei Fan (U. of Edinburgh)
Amélie Gheerbrant (LIAFA, U. Paris Diderot)
Giorgio Ghelli (U. di Pisa)
Benny Kimelfeld (Technion)
Markus Krötzsch (TU Dresden)
Carsten Lutz (U. Bremen)
Wim Martens (U. of Bayreuth)
Anca Muscholl (LaBRI)
Jeffrey Naughton (U. of Wisconsin)
Dan Olteanu (U. of Oxford)
Juan Reutter (PUC Chile)
Luc Segoufin (INRIA)
Dan Suciu (U. of Washington)
Dirk Van Gucht (Indiana U.)
Ke Yi (Hong Kong U. of Science and Technology)

Proceedings Chair

Thomas Zeume (TU Dortmund U.)

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume

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

External Reviewers

Pierre Bourhis
Simone Bova
Florent Capelli
Nicolas de Rugy-Altherre
André Hernich
Markus Latte
Stefan Mengel
Sebastian Siebertz
Francesco Silvestri
Domagoj Vrgoč
Haitao Wang
Adam Witkowski
Qirun Zhang

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume

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

List of Authors

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The ICDT 2016 Test of Time Award Announcement

Foto N. Afrati[1], Claire David[2], and Georg Gottlob[3]

1 **National Technical Universtity of Athens, Athens, Greece**
afrati@softlab.ece.ntua.gr
2 **Université Paris-Est Marne-la-Vallée, Marne-la-Vallée, France**
Claire.David@u-pem.fr
3 **University of Oxford, Oxford, UK**
georg.gottlob@cs.ox.ac.uk

Abstract

We describe the 2016 ICDT Test of Time Award which is awarded to Chandra Chekuri and Anand Rajaraman for their 1997 ICDT paper on "Conjunctive Query Containment Revisited".

1998 ACM Subject Classification H.2.3 [Database Management] Languages

Keywords and phrases conjunctive query, treewidth, NP-hardness, rewriting

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.1

1 The ICDT 2016 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 2016, the award recognizes a paper selected from the proceedings of the ICDT 1995 & 1997 conferences, 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 2016 Joint Conference, March 15–18, 2016 in Bordeaux, France.

The 2016 Test of Time Award Committee, consisting of Foto N. Afrati, Claire David, and Georg Gottlob (chair), has chosen the following contribution for the 2016 ICDT Test of Time Award:[1]

Conjunctive Query Containment Revisited
by Chandra Chekuri and Anand Rajaraman
6th International Conference on Database Theory (ICDT 1997)

The paper is available here: http://dx.doi.org/10.1007/3-540-62222-5_36.

2 Contribution

This landmark paper made highly significant contributions to the problems of conjunctive query containment and optimization. While it was known that these NP-hard problems are tractable in case of acyclic queries, Chekuri and Rajaraman observed that the most

[1] Full citation is given in [1].

commonly encountered queries, while not necessarily acyclic, are in some sense nearly acyclic and still lend themselves to polynomial-time containment and minimization algorithms. To make this precise, they introduced the concept of *query width*, which is based on the notion of *query decomposition* combining treewidth-like decomposition techniques with set covering methods. In particular, the class of acyclic queries coincides with the class of queries having query width 1. They showed that the problems of query-containment and query minimization are tractable for classes of queries whose query width is bounded by some constant k in case a query decomposition of width $\leq k$ is given.

The paper contains a number of further important results on (i) the relationship between the query width of a query and the treewidth of its incidence graph, (ii) the hardness of approximating query minimization, and (iii) rewriting and answering queries of bounded query width in presence of views.

This highly cited paper, whose full version has appeared in *Theoretical Computer Science* [2], had a major impact on subsequent work in Database Theory and Artificial Intelligence (in particular, constraint satisfaction). Its pioneering use of hypergraph-based rather than graph-based decomposition techniques marked the beginning of a still ongoing series of investigations that have led to the definition of further, successively more general decomposition techniques, rooted in the very idea of query decomposition.

Foto N. Afrati Claire David Georg Gottlob

The ICDT Test of Time Award Committee for 2016

References

1 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. In *Proc. of the 6th International Conference on Database Theory (ICDT'97), Delphi, Greece, January 8–10, 1997*, volume 1186 of *Lecture Notes in Computer Science*, pages 56–70. Springer, 1997. doi:10.1007/3-540-62222-5_36.

2 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.

Scale Independence: Using Small Data to Answer Queries on Big Data

Floris Geerts

Department of Mathematics & Computer Science, University of Antwerp, Belgium
floris.geerts@uantwerpen.be

Abstract

Large datasets introduce challenges to the scalability of query answering. Given a query Q and a dataset D, it is often prohibitively costly to compute the query answers $Q(D)$ when D is big. To this end, one may want to use heuristics, "quick and dirty" algorithms which return approximate answers. However, in many applications it is a must to find exact query answers. So, how can we efficiently compute $Q(D)$ when D is big or when we only have limited resources?

One idea is to find a small subset D_Q of D such that $Q(D_Q) = Q(D)$ where the size of D_Q is independent of the size of the underlying dataset D. Intuitively, when such a D_Q can be found for a query Q, the query is said to be *scale independent* [1, 2, 9]. Indeed, for answering such queries the size of the underlying database does not matter, i.e., query processing is independent of the scale of the database.

In this talk, I will survey various formalisms that enable large classes of queries to be scale independent. These formalisms primarily rely on the availability of access constraints, a combination of indexes and cardinality constraints, on the data [8, 9]. We will take a closer look at how, in the presence of such constraints, queries can often be compiled into efficient query plans that access a bounded amount data [6, 8], and how these techniques relate to query processing in the presence of access patterns [3, 4, 7]. Finally, we illustrate that scale independent queries are quite common in practice and that they indeed can be efficiently answered on big datasets when access constraints are present [5, 6].

1998 ACM Subject Classification H.2.4 [Database Management] Systems – Query Processing, H.2.3 [Database Management] Languages – Query Languages, H.2.2 [Database Management] Physical Design – Access methods

Keywords and phrases Scale independence, Access constraints, Query processing

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.2

Category Invited Talk

References

1 Michael Armbrust, Kristal Curtis, Tim Kraska, Armando Fox, Michael J. Franklin, and David A. Patterson. PIQL: Success-tolerant query processing in the cloud. *PVLDB*, 5(3):181–192, 2011.

2 Michael Armbrust, Eric Liang, Tim Kraska, Armando Fox, Michael J. Franklin, and David A. Patterson. Generalized scale independence through incremental precomputation. In *Proc SIGMOD 2013*, pages 625–636, 2013.

3 Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. Querying with access patterns and integrity constraints. *PVLDB*, 8(6):690–701, 2015.

4 Michael Benedikt, Balder ten Cate, and Efthymia Tsamoura. Generating low-cost plans from proofs. In *Proc. PODS 2014*, pages 200–211, 2014.

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume; Article No. 2; pp. 2:1–2:2

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

5 Yang Cao, Wenfei Fan, Jinpeng Huai, and Ruizhe Huang. Making pattern queries bounded in big graphs. In *Proc. ICDE 2015*, pages 161–172, 2015.

6 Yang Cao, Wenfei Fan, Tianyu Wo, and Wenyuan Yu. Bounded conjunctive queries. *PVLDB*, 7(12):1231–1242, 2014.

7 Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. *TCS*, 371(3):200–226, 2007.

8 Wenfei Fan, Floris Geerts, Yang Cao, Ting Deng, and Ping Lu. Querying big data by accessing small data. In *Proc. PODS 2015*, pages 173–184, 2015.

9 Wenfei Fan, Floris Geerts, and Leonid Libkin. On scale independence for querying big data. In *Proc. PODS 2014*, pages 51–62, 2014.

Top-k Indexes Made Small and Sweet

Yufei Tao

University of Queensland, Brisbane, Queensland, Australia
taoyf@itee.uq.edu.au

Abstract

Top-k queries have become extremely popular in the database community. Such a query, which is issued on a set of elements each carrying a real-valued *weight*, returns the k elements with the highest weights among all the elements that satisfy a predicate. As usual, an index structure is necessary to answer a query substantially faster than accessing the whole input set.

The existing research on top-k queries can be classified in two categories. The first one, which is system-oriented, aims to devise indexes that are simple to understand and easy to implement. These indexes, typically designed with heuristics, are reasonably fast in practical applications, but do not necessarily offer strong performance guarantees – in other words, they are *small* but not *sweet*. The other category, which is theory-oriented, aims to develop indexes that promise attractive bounds on the space consumption and query overhead (sometimes also update cost). These indexes, unfortunately, are often excessively sophisticated in the adopted techniques, and are rarely applied in practice – they are *sweet* but not *small*.

This talk will discuss the progress of an on-going project that strives to take down the barrier between the two categories, by crafting a framework for acquiring simple top-k indexes with excellent performance guarantees – namely, small and sweet. This is achieved with reductions that produce top-k indexes automatically from the existing data structures for conventional reporting queries on unweighted elements (i.e., finding all elements satisfying a predicate), and/or the existing data structures on top-1 queries. Our reductions promise nearly no performance deterioration with respect to those existing structures, are general enough to be applicable to a huge variety of top-k problems, and work in both the external memory model and the RAM model.

1998 ACM Subject Classification F.2.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Data Structures, Top-k, External Memory, RAM, Reductions

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.3

Category Invited Talk

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume; Article No. 3; pp. 3:1–3:1

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

New Algorithms for Heavy Hitters in Data Streams

David P. Woodruff

IBM Research Almaden, San Jose, CA, USA
dpwoodru@us.ibm.com

Abstract

An old and fundamental problem in databases and data streams is that of finding the heavy hitters, also known as the top-k, most popular items, frequent items, elephants, or iceberg queries. There are several variants of this problem, which quantify what it means for an item to be frequent, including what are known as the ℓ_1-heavy hitters and ℓ_2-heavy hitters. There are a number of algorithmic solutions for these problems, starting with the work of Misra and Gries, as well as the CountMin and CountSketch data structures, among others.

In this paper (accompanying an invited talk) we cover several recent results developed in this area, which improve upon the classical solutions to these problems. In particular, we develop new algorithms for finding ℓ_1-heavy hitters and ℓ_2-heavy hitters, with significantly less memory required than what was known, and which are optimal in a number of parameter regimes.

1998 ACM Subject Classification H.2.8 [DatabaseManagement] Database Applications – Data mining, C.2.3 [Computer-Communication Networks]: Network Operations – Network monitoring, F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems

Keywords and phrases data streams, heavy hitters

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.4

Category Invited Talk

1 The Heavy Hitters Problem

A well-studied problem in databases and data streams is that of finding the heavy hitters, also known as the top-k, most popular items, frequent items, elephants, or iceberg quries. These can be used for flow identification at IP routers [20], in association rules and frequent itemsets [1, 44, 47, 25, 24], and for iceberg queries and iceberg datacubes [21, 6, 23]. We refer the reader to the survey [16], which presents an overview of known algorithms for this problem, from both theoretical and practical standpoints.

There are various different flavors of guarantees for the heavy hitters problem. We start with what is known as the ℓ_1-guarantee:

▶ **Definition 1** (ℓ_1-(ϵ, ϕ)-Heavy Hitters Problem)**.** In the (ϵ, ϕ)-Heavy Hitters Problem, we are given parameters $0 < \epsilon < 1$ and $2\epsilon \leq \phi \leq 1$, as well as a stream $a_1, \ldots, a_m$ of items $a_j \in \{1, 2, \ldots, n\}$. Let f_i denote the number of occurrences of item i, i.e., its frequency. The algorithm should make one pass over the stream and at the end of the stream output a set $S \subseteq \{1, 2, \ldots, n\}$ for which if $f_i \geq \phi m$, then $i \in S$, while if $f_i \leq (\phi - \epsilon)m$, then $i \notin S$. Further, for each item $i \in S$, the algorithm should output an estimate $\tilde{f}_i$ of the frequency f_i which satisfies $|f_i - \tilde{f}_i| \leq \epsilon m$.

We are interested in algorithms which use as little space (i.e., memory) in bits as possible to solve the ℓ_1-(ϵ, ϕ)-Heavy Hitters Problem. We allow the algorithm to be randomized and

to succeed with probability at least $1-\delta$, for $0 < \delta < 1$. We do not make any assumption on the ordering of the stream $a_1, \ldots, a_m$. This is desirable, as often in applications one cannot assume a best-case or even a random order. We will assume m is known in advance, though many of the algorithms below (including ours) can deal with unknown m. We note that while the problem still makes sense for any $\phi > \epsilon$, it is well-known that an $\Omega(n)$ space lower bound exists when ϕ is very close to ϵ, e.g., if $\phi = \epsilon + 1/n$. Indeed, this follows via a reduction from communication complexity, which is a standard method for proving lower bounds in data streams. In particular, a reduction from the so-called INDEX problem is readily apparent - we refer the reader to [32] for more details of the communication problem (see, e.g., [43] for a recent survey discussing the INDEX problem).

The first algorithm for the ℓ_1-(ϵ, ϕ)-Heavy Hitters Problem was given by Misra and Gries [38], who achieved $O(\epsilon^{-1} \log n)$ bits of space for any $\phi > 2\epsilon$. This algorithm was rediscovered by Demaine et al. [18], and again by Karp et al. [31]. Other than these algorithms, which are deterministic, there are a number of randomized algorithms, such as the CountSketch [13], Count-Min sketch [17], sticky sampling [34], lossy counting [34], space-saving [36], sample and hold [20], multi-stage bloom filters [11], and sketch-guided sampling [33]. Berinde et al. [5] show that using $O(k\epsilon^{-1} \log(mn))$ bits of space, one can achieve the stronger guarantee of reporting, for each item $i \in S$, $\tilde{f}_i$ with $|\tilde{f}_i - f_i| \leq \frac{\epsilon}{k} F_1^{res(k)}$, where $F_1^{res(k)} < m$ denotes the sum of frequencies of items in $\{1, 2, \ldots, n\}$ excluding the frequencies of the k most frequent items. This is particularly useful when there are only a few large frequencies, since then the error $\frac{\epsilon}{k} F_1^{res(k)}$ will depend only on the remaining small frequencies.

While the ℓ_1-heavy hitters have a number of applications, there is also a sometimes stronger notion known as the ℓ_2-heavy hitters, which we now define.

▶ **Definition 2 (ℓ_2-(ϵ, ϕ)-Heavy Hitters Problem).** In the (ϵ, ϕ)-Heavy Hitters Problem, we are given parameters $0 < \epsilon < 1$ and $2\epsilon \leq \phi \leq 1$, as well as a stream $a_1, \ldots, a_m$ of items $a_j \in \{1, 2, \ldots, n\}$. Let f_i denote the number of occurrences of item i, i.e., its frequency. Let $F_2 = \sum_{i=1}^{n} f_i^2$. The algorithm should make one pass over the stream and at the end of the stream output a set $S \subseteq \{1, 2, \ldots, n\}$ for which if $f_i^2 \geq \phi F_2$, then $i \in S$, while if $f_i^2 \leq (\phi - \epsilon) F_2$, then $i \notin S$. Further, for each item $i \in S$, the algorithm should output an estimate $\tilde{f}_i$ of the frequency f_i which satisfies $|f_i - \tilde{f}_i| \leq \epsilon \sqrt{F_2}$.

One of the algorithms for ℓ_1-heavy hitters mentioned above, the CountSketch [14], refined in [46], actually solves the ℓ_2-(ϵ, ϕ)-Heavy Hitters Problem. Notice that this guarantee can be significantly stronger than the aforementioned ℓ_1-guarantee that $f_i \geq \epsilon m$. Indeed, if $f_i \geq \phi m$, then $f_i^2 \geq \phi^2 m^2 \geq \phi^2 F_2$. So, an algorithm for finding the ℓ_2-heavy hitters, with ϕ replaced by ϕ^2, will find all items satisfying the ℓ_1-guarantee with parameter ϕ. On the other hand, given a stream of n distinct items in which $f_{i^*} = \sqrt{n}$ for an $i^* \in [n] = \{1, 2, 3, \ldots, n\}$, yet $f_i = 1$ for all $i \neq i^*$, an algorithm satisfying the ℓ_2-heavy hitters guarantee will identify item i with constant ϕ, but an algorithm which only has the ℓ_1-guarantee would need to set $\phi = 1/\sqrt{n}$, therefore using $\Omega(\sqrt{n})$ bits of space. In fact, ℓ_2-heavy hitters are in some sense the best one can hope for with a small amount of space in a data stream, as it is known for $p > 2$ that finding those i for which $f_i^p \geq \phi F_p$ requires $n^{1-2/p}$ bits of space even for constant ϕ [4, 12].

The ℓ_2-heavy hitter algorithms of [14, 46] have broad applications in compressed sensing [22, 42, 37] and numerical linear algebra [15, 35, 40, 9], and are often used as a subroutine in other data stream algorithms, such as ℓ_p-sampling [39, 3, 29], cascaded aggregates [28], and frequency moments [27, 8].

Given the many applications of heavy hitters, it is natural to ask what the best space complexity for them is. For simplicity of presentation, we make the common assumption that the stream length m is polynomially related to the universe size n.

It is clear that for constant ϵ and ϕ, that there is an $\Omega(\log n)$ bit lower bound, as this is just the number of bits needed to specify the identity of the heavy hitter.

For constant ϵ, given the aforementioned results, this is actually tight for the ℓ_1-(ϵ, ϕ)-Heavy Hitters Problem. The main focus then, for the ℓ_1-(ϵ, ϕ)-Heavy Hitters Problem is on obtaining tight bounds as a function of ϵ and ϕ.

On the other hand, for the ℓ_2-(ϵ, ϕ)-Heavy Hitters Problem, even for constant ϵ and ϕ, the best previous algorithms of [14] and the followup [46] achieve $\Theta(\log^2 n)$ bits of space. It is known that if one allows deletions in the stream, in addition to insertions, then $\Theta(\log^2 n)$ bits of space is optimal [19, 29]. However, in many cases we just have a stream of insertions, such as in the model studied in the seminal paper of Alon, Matias, and Szegedy [2]. Thus, for the ℓ_2-(ϵ, ϕ)-Heavy Hitters Problem, our focus will be on the regime of constant ϵ and ϕ and on understanding the dependence on n.

There are a number of other desirable properties one would want out of a heavy hitters algorithm. For instance, one is often also interested in minimizing the *update time* and *reporting time* of such algorithms. Here, the update time is defined to be the time the algorithm needs to update its data structure when processing a stream insertion. The reporting time is the time the algorithm needs to report the answer after having processed the stream. In this article we will focus primarily on the space complexity.

2 Our Recent Results

In several recent works [10, 7], together with coauthors we significantly improve known algorithms for finding both ℓ_1-heavy hitters as well as ℓ_2-heavy hitters. For many settings of parameters, our algorithms are optimal.

2.1 ℓ_1-Heavy Hitters

In joint work with Bhattacharyya and Dey [7], we improve upon the basic algorithm of Misra and Gries [38] for the ℓ_1-(ϵ, ϕ)-Heavy Hitters Problem, the latter achieving $O(\epsilon^{-1} \log n)$ bits of space for any $\phi \geq 2\epsilon$. We now describe the algorithm of [7].

We first recall the algorithm of Misra and Gries. That algorithm initializes a table of $1/\epsilon + 1$ pairs of (v, c) to $(\perp, 0)$, where v is an element in the universe $\{1, 2, \ldots, n\} \cup \perp$, and c is a non-negative integer. When receiving a new stream insertion a_i, the algorithm checks if $v = a_i$ for some (v, c) pair in the table. If so, it replaces (v, c) with $(v, c + 1)$. Otherwise, if there is a (v, c) in the table with $v = \perp$, then the algorithm replaces that (v, c) pair with $(a_i, 1)$. If neither of the previous two cases hold, the algorithm takes each (v, c) pair in the table, and replaces it with $(v, c - 1)$. If $c - 1 = 0$, then the corresponding v is replaced with $\perp$.

Note that the algorithm, as described in the previous paragraph, naturally can be implemented using $O(\epsilon^{-1} \log n)$ bits of space (recall we assume the stream length m and the universe size n are polynomially related, so $\log m = \Theta(\log n)$). Moreover, a nice property is that the algorithm is deterministic.

For the correctness, note that if an item i occurs $f_i \geq 2\epsilon m$ times, then it will appear in the table at the end of the stream. Indeed, notice that for each occurrence of i in the stream, if it is not included in the table via the operation of replacing a pair (i, c) with $(i, c + 1)$ for some value of c, or replacing a pair $(\perp, 0)$ with $(i, 1)$, then this means that there were at least $1/\epsilon + 1$ stream updates that were removed from the table upon seeing this occurrence of i, since each counter c for each (v, c) pair in the table is decremented by 1. We can therefore charge those stream updates to this occurrence of i. Moreover, if (i, c) is in the table for

some value of c and is replaced with $(i, c-1)$ or $(\perp, 0)$, this means we can charge at least $1/\epsilon$ stream updates to items not equal to i to this occurrence of i. Since we are charging distinct stream updates for each occurrence of i, we have the relationship that $f_i \cdot (1/\epsilon) \leq m$, which is a contradiction to $f_i \geq 2\epsilon m$. Therefore, i will occur in a pair in the table at the end of the stream. The same analysis in fact implies that at most ϵm occurrences of i will not be accounted for in the table at the end of the stream, which means that for the (i, c) pair in the table, we have $f_i \geq c \geq f_i - \epsilon m$. This latter guarantee enables us to solve the ℓ_1-(ϵ, ϕ)-Heavy Hitters Problem for any $\phi \geq 2\epsilon$.

One shortcoming of the algorithm above is that if ϕ is much larger than ϵ, say ϕ is constant, then the above algorithm still requires $O(\epsilon^{-1} \log n)$ bits of space, that is, it is insensitive to the value of ϕ. Consider for instance, the case when $\epsilon = 1/\log n$ and $\phi = 1/10$, so one wants a very high accuracy estimate to each of the item frequencies for items occurring at least 10% of the time. The above algorithm would use $O(\log^2 n)$ bits of space for this problem. In this case, the only known lower bound is $\Omega(\log n)$ bits, which just follows from the need to return the identities of the heavy hitters. Is it possible to improve this $O(\log^2 n)$ bits of space upper bound?

This is precisely what we show in [7]. Here we sketch how to achieve a bound of $O((1/\phi) \log n + (1/\epsilon) \log(1/\epsilon))$ bits of space and refer to [7] for further optimizations as well as extensions to related problems. Note that this translates to a space bound of $O(\log n \log \log n)$ bits for the above setting of parameters.

The first observation is that if we randomly sample $r = \Theta(1/\epsilon^2)$ stream updates, then with probability 99%, simultaneously for every universe item i, if we let $\hat{f}_i$ denote its frequency among the samples, and f_i its frequency in the original stream, then we have

$$\left| \frac{\hat{f}_i}{r} - \frac{f_i}{m} \right| \leq \frac{\epsilon}{2}.$$

This follows by Chebyshev's inequality and a union bound. Indeed, consider a given $i \in [n]$ with frequency f_i and suppose we sample each of its occurrences pairwise-independently with probability r/m, for a parameter r. Recall that pairwise independence here implies that any single occurrence is sampled with probability r/m and any two occurrences are jointly sampled with probability exactly r^2/m^2, though we do not impose any constraints on the joint distribution of any three or more samples. Also, a pairwise independent hash function can be represented with only $O(\log n)$ bits of space. Then the expected number $\mathbf{E}[\hat{f}_i]$ of sampled occurrences is $f_i \cdot r/m$ and the variance $\mathbf{Var}[\hat{f}_i]$ is $f_i \cdot r/m(1 - r/m) \leq f_i r/m$ (here we use pairwise independence to conclude the same variance bound as if the samples were fully independent). Applying Chebyshev's inequality,

$$\Pr\left[\left|\hat{f}_i - \mathbf{E}[\hat{f}_i]\right| \geq \frac{r\epsilon}{2}\right] \leq \frac{\mathbf{Var}[\hat{f}_i]}{(r\epsilon/2)^2} \leq \frac{4 f_i r}{m r^2 \epsilon^2}.$$

Setting $r = \frac{C}{\epsilon^2}$ for a constant $C > 0$ makes this probability at most $\frac{4 f_i}{Cm}$. By the union bound, if we sample each element in the stream independently with probability $\frac{r}{m}$, then the probability there exists an i for which $|\hat{f}_i - \mathbf{E}[\hat{f}_i]| \geq \frac{r\epsilon}{2}$ is at most $\sum_{i=1}^{n} \frac{4 f_i}{Cm} \leq \frac{4}{C}$, which for $C \geq 400$ is at most $\frac{1}{100}$, as desired.

After sampling so that the stream length is reduced to $O(1/\epsilon^2)$, it follows that the number of distinct items in the stream is also $O(1/\epsilon^2)$, and therefore if we hash the item identifiers to a universe of size $O(1/\epsilon^4)$, by standard arguments with probability 99% the items will be perfectly hashed, that is, there will be no collisions. This follows even with a pairwise-independent hash function h. The high level idea then is to run the algorithm of

Misra and Gries, but the pairs (v, c) correspond to the *hashed* item identity and the *count in the sampled stream*, respectively. Notice that it takes only $O(\log(1/\epsilon))$ bits to represent such pairs and so the algorithm of Misra and Gries would take $O(\epsilon^{-1}\log(1/\epsilon))$ bits of space.

However, we still want to return the actual item identifiers! To do this, we maintain a parallel data structure containing actual item identifiers in $[n]$, but the data structure only contains $O(1/\phi)$ items. In particular, these item identities correspond to the items v for which $(h(v), c)$ is stored in the algorithm of Misra and Gries, for which the c values are largest. Namely, the items with top $1/\phi$ c-values have their actual identities stored. This can be maintained under stream insertions since given a new stream update, one has the actual identity in hand, and therefore can appropriately update the identities of the items with top $O(1/\phi)$ counts. Moreover, when we subtract one from all counters in the algorithm of Misra and Gries, the only thing that changes in the top $O(1/\phi)$ identities is that some of them may now have zero frequency, and so can be thrown out. Thus, we can always maintain the actual top $O(1/\phi)$ identities in the original (before hashing) universe.

We refer the reader to [7] for more details, optimizations, and extensions to related problems.

2.2 ℓ_2-Heavy Hitters

In joint work with Braverman, Chestnut, and Ivkin [10], we improve upon the CountSketch data structure [14] for the ℓ_2-(ϵ, ϕ)-Heavy Hitters Problem. To illustrate the algorithm of [10], we consider ϵ and ϕ to be constants in what follows, and further, we suppose there is only a single $i^* \in [n]$ for which $f_{i^*}^2 \geq \phi F_2$ and there is no i for which $(\phi - \epsilon)F_2 \leq f_i^2 < \phi F_2$. It is not hard to reduce to this case by first hashing into $O(1)$ buckets (recall ϕ, ϵ are constants for this discussion), since the $O(1/\phi)$ heavy hitters will go to separate buckets with large constant probability (if, say, we have $\Omega(1/\phi^2)$ buckets). Thus, we focus on this case. In this case the CountSketch algorithm would use $\Theta(\log^2 n)$ bits of space, whereas in [10] we achieve $O(\log n \log\log n)$ bits of space, nearly matching the trivial $\Omega(\log n)$ bit lower bound.

We first explain the CountSketch data structure. The idea is to assign each item $i \in [n]$ a random sign $\sigma(i) \in \{-1, 1\}$. We also randomly partition $[n]$ into B buckets via a hash function h and maintain a counter $c_j = \sum_{i|h(i)=j} \sigma(i) \cdot f_i$ in the j-th bucket. Then, to estimate any given frequency f_i, we estimate it as $\sigma(i) \cdot c_{h(i)}$. Note that $\mathbf{E}[\sigma(i) \cdot c_{h(i)}] = \mathbf{E}[\sigma(i)^2 f_i + \sum_{j \neq i, h(j)=h(i)} f_j \sigma(j)\sigma(i)] = f_i$, using that $\mathbf{E}[\sigma(i)\sigma(j)] = 0$ for $i \neq j$. Moreover, by computing the variance and applying Chebyshev's inequality, one has that

$$|\sigma(i) \cdot c_{h(i)} - f_i| = O(\sqrt{F_2/B})$$

with probability at least $9/10$. The intuitive explanation is that due to the random sign combination of remaining items in the same hash bucket as i, the absolute value of this linear combination concentrates to the Euclidean norm of the frequency vector of these items. The idea then is to repeat this independently $O(\log n)$ times in parallel. Then we estimate f_i by taking the median of the estimates across each of the $O(\log n)$ repetitions. By Chernoff bounds, we have that with probability $1 - 1/n^2$, say, the resulting estimate is within an additive $O(\sqrt{F_2/B})$ of the true frequency f_i. This then holds for every $i \in [n]$ simultaneously by a union bound, at which point one can then find the ℓ_2-heavy hitters, if say, one sets $B = \Theta(1/\epsilon^2)$.

Notice that it is easy to maintain the CountSketch data structure in a data stream since we just need to hash the new item i to the appropriate bucket and add $\sigma(i)$ to the counter in that bucket, once for each of the $O(\log n)$ repetitions. The total space complexity of the CountSketch algorithm is $O(B \cdot \log^2 n)$, where the "B" is the number of hash buckets,

one $\log n$ factor is to store the counter in each bucket, and the other $\log n$ factor is for the number of repetitions. For constant ϵ and $B = \Theta(1/\epsilon^2)$ this gives $O(\log^2 n)$ bits of space. It is also not hard to see that the CountSketch data structure can be maintained in a stream with deletions as well as insertions, since given a deletion to item i, this just corresponds to subtracting $\sigma(i)$ from the bucket i hashes to in each repetition. Moreover, as mentioned earlier, this $O(\log^2 n)$ space bound is optimal for streams with deletions.

To give some intuition for our new algorithm, let $i^* \in [n]$ be the identity of the single ℓ_2-heavy hitter that we wish to find. Suppose first that $f_{i^*} \geq \sqrt{n} \log n$ and that $f_i \in \{0, 1\}$ for all $i \in [n] \setminus \{i^*\}$. For the moment, we are also going to ignore the issue of storing random bits, so assume we can store $\text{poly}(n)$ random bits for free (which can be indexed into using $O(\log n)$ bits of space). We will later sketch how to remove this assumption. As in the CountSketch algorithm, we again assign a random sign $\sigma(i)$ to each item $i \in [n]$. Suppose we randomly partition $[n]$ into two buckets using a hash function $h : [n] \to \{1, 2\}$, and correspondingly maintain two counters $c_1 = \sum_{i|h(i)=1} \sigma(i) \cdot f_i$ and $c_2 = \sum_{i|h(i)=2} \sigma(i) \cdot f_i$. Suppose for discussion that $h(i^*) = 1$. A natural question is what the values c_1 and c_2 look like as we see more updates in the stream.

Consider the values $c_1 - \sigma(i^*) \cdot f_{i^*}$ and c_2. Then, since all frequencies other than i^* are assumed to be 0 or 1, and since the signs $\sigma(j)$ are independent, these two quantities evolve as random walks starting at 0 and incrementing by $+1$ with probability $1/2$, and by -1 with probability $1/2$, at each step of the walk. By standard theory of random walks (e.g., Levy's theorem), there is a constant $C > 0$ so that with probability at least $9/10$, simultaneously at all times during the stream we have that $|c_1 - \sigma(i^*) \cdot f_{i^*}|$ and $|c_2|$ are upper bounded by $C\sqrt{n}$. The constant of $9/10$, like typical constants in this paper, is somewhat arbitrary. This suggests the following approach to learning i^*: at some point in the stream we will have that $f_{i^*} > 2C\sqrt{n}$, and at that point $|c_1| > C\sqrt{n}$, but then we know that i^* occurs in the first bucket. This is assuming that the above event holds for the random walks. Since we split $[n]$ randomly into two pieces, this gives us 1 bit of information about the identity of i^*. If we were to repeat this $O(\log n)$ times in parallel, we would get exactly the CountSketch data structure, which would use $\Theta(\log^2 n)$ bits of space. Instead, we get much better space by repeating $\Theta(\log n)$ times sequentially!

To repeat this sequentially, we simply wait until either $|c_1|$ or $|c_2|$ exceeds $Cn^{1/2}$, at which point we learn one bit of information about i^*. Then, we reset the two counters to 0 and perform the procedure again. Assuming $f_{i^*} = \Omega(\sqrt{n} \log n)$, we will have $\Omega(\log n)$ repetitions of this procedure, each one succeeding independently with probability $9/10$. By Chernoff bounds, there will only be a single index $i \in [n]$ which match a $2/3$ fraction of these repetitions, and necessarily $i = i^*$.

2.2.1 Gaussian Processes

In general we do not have $f_{i^*} = \Omega(\sqrt{n} \log n)$, nor do we have that $f_i \in \{0, 1\}$ for all $i \in [n] \setminus \{i^*\}$. We fix both problems using the theory of Gaussian processes.

▶ **Definition 3.** A **Gaussian process** is a collection $\{X_t\}_{t \in T}$ of random variables, for an index set T, for which every finite linear combination of the random variables is Gaussian.

We assume $\mathbf{E}[X_t] = 0$ for all t, as this will suffice for our application. It then follows that the Gaussian process is entirely determined by its covariances $\mathbf{E}[X_s X_t]$. This fact is related to the fact that a Gaussian distribution is determined by its mean and covariance. The distance function $d(s, t) = (\mathbf{E}[(X_s - X_t)^2])^{1/2}$ is then a pseudo-metric on T (the only property it lacks of a metric is that $d(s, t)$ may equal 0 if $s \neq t$).

The connection to data streams is the following. Suppose we replace the signs $\sigma(i)$ with standard normal random variables $g(i)$ in our counters above, and consider a counter c at time t, denoted $c(t)$, of the form $\sum_i g(i) \cdot f_i(t)$. Here $f_i(t)$ is the frequency of item i after processing t stream insertions. The main point is that $c(t)$ is a Gaussian process! Indeed, any linear combination of the $c(t)$ values for different t is again Gaussian since the sum of normal random variables is again a normal random variable.

The reason we wish to make such a connection to Gaussian processes is the following powerful inequality called the "chaining inequality".

▶ **Theorem 4** (Talagrand [45]). *Let $\{X_t\}_{t\in T}$ be a Gaussian process and let $T_0 \subseteq T_1 \subseteq T_2 \subseteq \cdots \subseteq T$ be such that $|T_0| = 1$ and $|T_i| \leq 2^{2^i}$ for $i \geq 1$. Then,*

$$\boldsymbol{E}\left[\sup_{t\in T} X_t\right] \leq O(1) \cdot \sup_{t\in T} \sum_{i\geq 0} 2^{i/2} d(t, T_i),$$

where $d(t, T_i) = \min_{s\in T_i} d(t, s)$.

We wish to apply Theorem 4 to the problem of finding ℓ_2-heavy hitters. Let $F_2(t)$ be the value of the second moment F_2 after seeing t stream insertions. We now describe how to choose the sets T_i in order to apply the chaining inequality; the intuition is that we recursively partition the stream based on its F_2 value.

Let a_t be the first stream update for which $F_2(m)/2 \leq F_2(t)$. Then $T_0 = \{t\}$. We then let T_i be the set of 2^{2^i} times $t_1, t_2, \ldots, t_{2^{2^i}}$ in the stream for which t_j is the first point in the stream for which $j \cdot F_2(m)/2^{2^i} \leq F_2(t_j)$. Then, we have created a nested sequence of subsets $T_0 \subseteq T_1 \subseteq T_2 \subseteq \cdots \subseteq T$ with $|T_0| = 1$ and $|T_i| \leq 2^{2^i}$ for $i \geq 1$.

We are now in position to apply Theorem 4. A straightforward computation based on our recursive partitioning of the stream around where F_2 changes (see [10] for details) shows that for any stream position t and set T_i we have created,

$$d(t, T_i) = \left(\mathbf{E}[\min_{s\in T_i} |c(t) - c(s)|^2]\right)^{1/2} = O\left(\frac{F_2}{2^{2^i}}\right)^{1/2}.$$

Applying Theorem 4, we have

$$\mathbf{E}[\sup_{t\in T} X_t] \leq O(1) \sup_{t\in T} \sum_{i\geq 0} 2^{i/2} \left(\frac{F_2}{2^{2^i}}\right)^{1/2} = O(F_2^{1/2}).$$

This is exactly the same bound that the theory for random walks gave us earlier! (recall in that case $\sum_{i\neq i^*} f_i^2 < n$).

Using Gaussian processes has therefore allowed us to remove our earlier assumption that $f_i \in \{0, 1\}$ for all $i \in [n] \setminus \{i^*\}$. The same random walk based algorithm will now work; however, we still need to assume the $f_{i^*} = \Omega(\sqrt{F_2} \log n)$ in order to learn $\log n$ bits of information to identify i^*, as before. This is not satisfactory, as an ℓ_2-heavy hitter only satisfies $f_i = \Omega(\sqrt{F_2})$ (recall we have assumed ϕ and ϵ are constants), which is weaker than the $f_{i^*} = \Omega(\sqrt{F_2} \log n)$ that the above analysis requires.

2.2.2 Amplification

To remove the assumption that $f_{i^*} = \Omega(\sqrt{F_2} \log n)$, our work [10] designs what we call an "amplification" procedure. This involves for $j = 1, 2, \ldots, O(\log \log n)$, independently choosing a pairwise independent hash function $h^j : [n] \to \{1, 2\}$. For each j, we as before

maintain two counters $c_1^j = \sum_{i|h^j(i)=1} g_j(i) \cdot f_i$ and $c_2^j = \sum_{i|h^j(i)=2} g_j(i) \cdot f_i$, where the $g_j(i)$ are independent standard normal random variables.

Applying the chaining inequality to each of the $O(\log \log n)$ counters created, we have that with large constant probability, in a constant fraction of the $O(\log \log n)$ pairs, both counters c_1^j and c_2^j will be bounded by $O(\sqrt{F_2})$ in magnitude. It follows that if $f_{i^*} \geq C\sqrt{F_2}$ for a sufficiently large constant $C > 0$ (which we can assume by first hashing the universe into $O(1)$ buckets before the streaming algorithm begins), then in say, a $9/10$ fraction of pairs j, the counter c_k^j, $k \in \{1, 2\}$, of larger magnitude will contain i^*. Moreover, by Chernoff bounds, only a $\frac{1}{\log^c n}$ fraction of other $i \in [n]$ will hash to the larger counter in at least a $9/10$ fraction of such pairs, where $c > 0$ is a constant that can be made arbitrarily large by increasing the constant in the number $O(\log \log n)$ of pairs of counters created. Now the idea is to effectively run our previous algorithm only on items which hash to the heavier counter in at least a $9/10$ fraction of pairs. By definition, this will contain i^*, and now the expected second moment of the other items for which we run the algorithm on will be $F_2/\log^c n$, which effectively makes $f_{i^*} = \Omega(\sqrt{F_2} \log n)$, where F_2 is now measured with respect to the items for which we run the algorithm on. Now we can sequentially learn $O(\log n)$ bits of information about i^* in our algorithm, as before.

One thing to note about this approach is that after seeing a sufficiently large number of insertions of i^*, i.e., $\Theta(\sqrt{F_2})$ such insertions, then most of the pairs of counters will have the property that the larger counter (in absolute value) stays larger forever. This is due to the chaining inequality. This can be used to fix the itemset for which we run the algorithm on. In fact, this is precisely why this does not result in a 2-pass algorithm, which one might expect since one does not know the itemset to run our algorithm on in advance. However, we always run the algorithm on whichever current itemset agrees with at least a $9/10$ fraction of the larger counters, and just accept the fact that in the beginning of the stream the bits we learn about i^* are nonsense; however, after enough updates to i^* have occurred in the stream then the counters "fix" themselves in the sense that the larger counter does not change. At this point the bits we learn about i^* in our algorithm are the actual bits that we desire. At the end of the stream, we only look at a suffix of these bits to figure out i^*, thereby ignoring the nonsensical bits at the beginning of the stream. We refer the reader to [10] for more details.

2.2.3 Derandomization

The final piece of the algorithm is to account for the randomness used by the algorithm. We need to derandomize the counters, which use the theory of Gaussian processes to argue their correctness. We also cannot afford to maintain all of the hash functions that were used to learn specific bits of i^* (which we need ad the end of the stream to figure out what i^* is).

To derandomize the Gaussian processes, we use a derandomized Johnson Lindenstrauss transform of Kane, Meka, and Nelson [30]. The rough idea is to first apply a Johnson-Lindenstrauss transform to the frequency vectors for which we take inner products with independent Gaussian random variables in our counters. This will reduce the dimension from n to $O(\log n)$, for which we can then afford to take an inner product with fully independent Gaussian random variables. The nice thing about Johnson-Lindenstrauss transforms is that they preserve all the covariances up to a constant factor in our specific Gaussian process, and therefore we can use Slepian's Lemma (see [10] for details) to argue that the Gaussian process is roughly the same as before, since it is entirely determined by its covariances. Here the derandomized Johnson-Lindenstrauss transform of [30] can be represented using only $O(\log n \log \log n)$ bits of space. Also, instead of using Gaussian random variables, which

require truncation, we can directly use sign random variables (+1 with probability 1/2, −1 with probability 1/2), which results in what are called Bernoulli processes, together with a comparison theorem for Bernoulli processes and Gaussian processes. This enables us to avoid arguments about truncating Gaussians.

To derandomize the hash functions, we use Nisan's pseudorandom generator in a similar way that Indyk uses it for derandomizing his algorithms for norm estimation [41, 26]. Please see [10] for further details.

3 Conclusions

We presented new algorithms for finding ℓ_1-heavy hitters and ℓ_2-heavy hitters in a data stream. We refer the reader to the original papers [10, 7] for further details. As these algorithms are inspired from applications in practice, it is very interesting to see how the improved theoretical algorithms perform in practice. In ongoing work we are testing these algorithms in practice on real datasets.

Another interesting aspect is that the technique of using Gaussian processes in the ℓ_2-heavy hitters algorithm has led to a number of other improvements to data stream algorithms, including for example the ability to estimate the second moment F_2 at all times in a stream of insertions. Previously, given a stream of length n and a universe of size n, to estimate F_2 at all points in a stream up to a constant factor would require $\Theta(\log^2 n)$ bits of space, since it takes $\Theta(\log n \log(1/\delta))$ bits to estimate it at a single point with failure probability δ, and one needs to union bound over n stream positions. Using Gaussian processes, we achieve only $O(\log n \log\log n)$ bits of space for this task. It would be interesting to see if Gaussian processes are useful for other problems in data streams.

Obtaining simultaneously optimal update time, reporting time, and space in all parameter regimes is also a very important goal.

References

1 Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

3 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 363–372, 2011.

4 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

5 Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.*, 35(4):26, 2010. doi:10.1145/1862919.1862923.

6 Kevin S. Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA.*, pages 359–370, 1999.

7 Arnab Bhattacharyya, Palash Dey, and David P. Woodruff. A new algorithm for ℓ_1-heavy hitters in insertion streams and related problems. *Manuscript*, 2016.

8 Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *Proceedings of the*

Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006, pages 708–713, 2006.

9 Jean Bourgain and Jelani Nelson. Toward a unified theory of sparse dimensionality reduction in euclidean space. *CoRR*, abs/1311.2542, 2013.

10 Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, and David P. Woodruff. Beating countsketch for heavy hitters in insertion streams. *CoRR*, abs/1511.00661, 2015.

11 Yousra Chabchoub, Christine Fricker, and Hanene Mohamed. Analysis of a bloom filter algorithm via the supermarket model. In *21st International Teletraffic Congress, ITC 2009, Paris, France, September 15-17, 2009*, pages 1–8, 2009.

12 Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 107–117, 2003.

13 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

14 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

15 Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 81–90, 2013.

16 Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

17 Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

18 Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Algorithms—ESA 2002*, pages 348–360. Springer, 2002.

19 Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. Lower bounds for sparse recovery. *CoRR*, abs/1106.0365, 2011.

20 Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.

21 Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 299–310, 1998.

22 Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. Approximate sparse recovery: optimizing time and measurements. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 475–484, 2010.

23 Jiawei Han, Jian Pei, Guozhu Dong, and Ke Wang. Efficient computation of iceberg cubes with complex measures. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 1–12, 2001.

24 Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 1–12, 2000.

25 Christian Hidber. Online association rule mining. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA.*, pages 145–156, 1999.

26 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

27 Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 202–208, 2005.

28 T. S. Jayram and David P. Woodruff. The data stream space complexity of cascaded norms. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 765–774, 2009.

29 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58, 2011.

30 Daniel M. Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit johnson-lindenstrauss families. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 628–639, 2011.

31 Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 28(1):51–55, 2003.

32 Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

33 Abhishek Kumar and Jun (Jim) Xu. Sketch guided sampling – using on-line estimates of flow size for adaptive data collection. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 23-29 April 2006, Barcelona, Catalunya, Spain*, 2006.

34 Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

35 Xiangrui Meng and Michael W. Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 91–100, 2013.

36 Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th International Conference on Database Theory*, ICDT'05, pages 398–412, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-30570-5_27.

37 Gregory T. Minton and Eric Price. Improved concentration bounds for count-sketch. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 669–686, 2014.

38 Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

39 Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error lp-sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1143–1160, 2010.

40 Jelani Nelson and Huy L. Nguyen. OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 117–126, 2013.

41 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

42 Eric Price. Efficient sketches for the set query problem. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 41–56, 2011.

43 Tim Roughgarden. Communication complexity (for algorithm designers). *CoRR*, abs/1509.06257, 2015.

44 Ashok Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland.*, pages 432–444, 1995.

45 Michel Talagrand. Majorizing measures: The generic chaining. *The Annals of Probability*, 24(3), 1996.

46 Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.

47 Hannu Toivonen. Sampling large databases for association rules. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 134–145, 1996.

Beyond Well-designed SPARQL

Mark Kaminski[1] and Egor V. Kostylev[2]

1 **Department of Computer Science, University of Oxford, UK**
2 **Department of Computer Science, University of Oxford, UK**

Abstract

SPARQL is the standard query language for RDF data. The distinctive feature of SPARQL is the OPTIONAL operator, which allows for partial answers when complete answers are not available due to lack of information. However, optional matching is computationally expensive – query answering is PSPACE-complete. The well-designed fragment of SPARQL achieves much better computational properties by restricting the use of optional matching – query answering becomes coNP-complete. However, well-designed SPARQL captures far from all real-life queries – in fact, only about half of the queries over DBpedia that use OPTIONAL are well-designed.

In the present paper, we study queries outside of well-designed SPARQL. We introduce the class of weakly well-designed queries that subsumes well-designed queries and includes most common meaningful non-well-designed queries: our analysis shows that the new fragment captures about 99% of DBpedia queries with OPTIONAL. At the same time, query answering for weakly well-designed SPARQL remains coNP-complete, and our fragment is in a certain sense maximal for this complexity. We show that the fragment's expressive power is strictly in-between well-designed and full SPARQL. Finally, we provide an intuitive normal form for weakly well-designed queries and study the complexity of containment and equivalence.

1998 ACM Subject Classification H.2.3 Languages – Query languages

Keywords and phrases RDF, Query languages, SPARQL, Optional matching

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.5

1 Introduction

The Resource Description Framework (RDF) [29, 17, 21] is the W3C standard for representing linked data on the Web. RDF models information in terms of labeled graphs consisting of triples of resource identifiers (IRIs). The first and last IRIs in such a triple, called *subject* and *object*, represent entity resources, while the middle IRI, called *predicate*, represents a relation between the two entities.

SPARQL [35, 20] is the default query language for RDF graphs. First standardised in 2008 [35], SPARQL is now recognised as a key technology for the Semantic Web. This is witnessed by a recently adopted new version of the standard, SPARQL 1.1 [20], as well as by active development of SPARQL query engines in academia and the industry, for instance, as part of the systems AllegroGraph [1], Apache Jena [2], Sesame [3], or OpenLink Virtuoso [4].

In recent years, SPARQL has been subject to a substantial amount of theoretical research, based on the foundational work by Pérez et al. [30, 31]. In particular, we now know much about evaluation [36, 28, 6, 32, 25, 23, 7, 22], optimisation [27, 33, 16, 15, 12, 24], federation [14, 13], expressive power [5, 34, 25, 39], and provenance tracking [18, 19] for queries from various fragments and extensions of SPARQL. These studies have had a great impact in the community, in fact influencing the evolution of SPARQL as a standard.

A distinctive feature of SPARQL as compared to SQL is the OPTIONAL operator (abbreviated as OPT in this paper). This operator was introduced to *"not reject (solutions)*

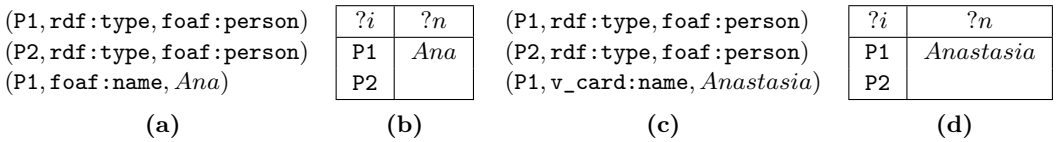


Figure 1 **(a)** Graph G; **(b)** answers to query (1) over G; **(c)** graph G'; and **(d)** answers over G'.

because some part of the query pattern does not match" [35]. For instance, consider the SPARQL query

$$\mathsf{SELECT}\ ?i, ?n\ \mathsf{WHERE}\ (?i, \texttt{rdf:type}, \texttt{foaf:person})\ \mathsf{OPT}\ (?i, \texttt{foaf:name}, ?n), \tag{1}$$

which retrieves all person IDs from the graph together with their names; names, however, are optional – if the graph does not contain information about the name of a person, the person ID is still retrieved but the variable $?n$ is left undefined in the answer. For instance, query (1) has two answers over the graph G in Figure 1(a), where the second answer is partial (see Figure 1(b)). However, if we extend G with a triple supplying a name for P2, the second answer will include this name.

The OPT operator accounts in a natural way for the open world assumption and the fundamental incompleteness of the Web. However, evaluating queries that use OPT is computationally expensive – Pérez et al. [31] showed PSPACE-completeness of SPARQL query evaluation, and Schmidt et al. [36] refined this result by proving PSPACE-hardness even for queries using no operators besides OPT. This is not surprising given that SPARQL queries are equivalent in expressive power to first-order logic queries, and translations in both directions can be done in polynomial time [5, 34, 25].

This spurred a search for restrictions on the use of OPT that would ensure lower complexity of query evaluation. It was also recognised that queries that are difficult to evaluate are often unintuive. For instance, they may produce less specified answers (i.e., answers with fewer bound variables) as the graph over which they are evaluated grows larger.

Perez et al. [31] introduced the *well-designed* fragment of SPARQL queries by imposing a syntactic restriction on the use of variables in OPT-expressions. Roughly speaking, each variable in the optional (i.e., right) argument of an OPT-expression should either appear in the mandatory (i.e., left) argument or be globally fresh for the query, i.e., appear nowhere outside of the argument. Well-designed queries have lower complexity of query evaluation – the problem is coNP-complete (provided all the variables in the query are selected). Moreover, such queries have a more intuitive behaviour than arbitrary SPARQL queries; in particular, they enjoy the monotonicity property that we observed for query (1): each partial answer over a graph can potentially be extended to undefined variables if the graph is completed with the missing information, and the more information we have the more specified are the answers. Well-designed queries can be efficiently transformed to an intuitive normal form allowing for a transparent graphical representation of queries as trees [27, 33]. Hence, many recent studies concentrate partially [27, 25, 23, 37, 38] or entirely [33] on well-designed queries.

Such a success of well-designed queries may lead to the impression that non-well-designed SPARQL queries are just a useless side effect of the early specification. But is this impression justified by the use of SPARQL in practice? To answer this question, a comprehensive analysis of real-life queries is required. We are aware of two works that analyse the distribution of operators in SPARQL queries asked over DBpedia [32, 9]. Both studies show that OPT is used in a non-negligible amount of practical queries. However, only Picalausa and

Vansummeren [32] go further and analyse how many of these queries are well-designed; and the result is quite interesting – well-designed queries make up only about half of all queries with OPT. In other words, well-designed queries are common, but by far not exclusive.

The main goal of this paper is to investigate SPARQL queries beyond the well-designed fragment. We wanted to see if the well-designedness condition could be extended so as to include most practical queries while preserving good computational properties. The main result of our study is very positive – we identified a new fragment of SPARQL queries, called *weakly well-designed* queries, that covers about 99% of queries over DBpedia and has the same complexity of query evaluation as the well-designed fragment. We also show that our fragment is in a sense maximal for this complexity.

We next describe our results and techniques in more detail. Our first step was to identify most typical real-life queries that are not well-designed. We analysed the USEWOD2013 [10] and USEWOD2014 [11] query logs for DBpedia 3.8 and 3.9 and found two interesting types of non-well-designed queries. The first type is exemplified by the following query:

$$\begin{array}{l}\mathsf{SELECT}\ ?i, ?n\ \mathsf{WHERE}\\ ((?i, \texttt{rdf:type}, \texttt{foaf:person})\ \mathsf{OPT}\ (?i, \texttt{foaf:name}, ?n))\ \mathsf{OPT}\ (?i, \texttt{v\_card:name}, ?n).\end{array} \tag{2}$$

This query is clearly not well-designed because variable $?n$, binding the name of a person, appears in two different unrelated optional parts. Let us analyse answers to this query over different graphs. On graph G in Figure 1(a) the result is exactly the same as for query (1), shown in Figure 1(b), simply because the IRI `v_card:name` is not present in G, and so cannot be matched against the second optional part of the query. Similarly, on graph G' in Figure 1(c), where the source of the name and the name itself are different, the result is as in Figure 1(d). In this case, the first optional part in the query does not match anything in the graph so the variable $?n$ is left unbound at this point; then the second optional is matched, and the variable is assigned with the name from `v_card`. More interestingly, query (2) evaluated over the graph $G \cup G'$ once again yields the result in Figure 1(b). Indeed, in this case, the first optional part has a match again and $?n$ is assigned the value *Ana*; then, this variable is already bound and there is no match for the second optional part that agrees with this value, meaning that the alternative `v_card` name is disregarded by the query. To summarise, query (2) is once again looking for person IDs and, optionally, their names. Now, however, names are collected from two different sources, `foaf` and `v_card`, where the first source is given preference over the second (maybe because it is considered more reliable or more informative, or for some other reason). In other words, if we know the `foaf` name of a person, it is returned as part of the answer regardless of his `v_card` name; however, if there is no `foaf` name, then the `v_card` name is also acceptable and should be returned; variable $?n$ is left unbound only if the name cannot be extracted from either source.

Of course, preference patterns encountered in real-life queries are often more complex. Still, in most cases they do not increase the complexity of query evaluation.

Our second example query is as follows:

$$\begin{array}{l}\mathsf{SELECT}\ ?i, ?n\ \mathsf{WHERE}\ ((?i, \texttt{rdf:type}, \texttt{foaf:person})\ \mathsf{OPT}\ (?i, \texttt{foaf:name}, ?n))\\ \qquad\qquad \mathsf{FILTER}\ (\neg \mathit{bound}(?n) \vee \neg(?n = \mathit{Ana})).\end{array} \tag{3}$$

The query uses FILTER, a standard SPARQL operator that admits only answers conforming to a specified constraint. Again, this query is not well-designed because the FILTER constraint mentions the variable $?n$, which occurs in the optional part of the query but not in the mandatory part. However, the intention of the query is quite clear: it searches for people whose names are not known to be *Ana*, including people whose names are unknown.

This use of FILTER is in fact very common in real-life queries. Moreover, it is intuitive as long as FILTER is essentially the outermost operator in the query, as it is in our example. In all such cases, however, FILTER cannot lead to an increase in complexity.

Having isolated these typical uses of non-well-designedness, we identify a new fragment of SPARQL that (a) includes all queries of the above two types, (b) subsumes well-designed queries, and (c) has the same complexity of query evaluation as well-designed queries. We call such queries *weakly well-designed*. They are the maximal fragment without structural restrictions on conjunctive blocks and filter conditions that has the above properties. Our analysis shows that about 99% of DBpedia queries with OPT are weakly well-designed.

Besides low complexity of query evaluation, we establish a few more useful properties of weakly well-designed queries, which are summarised in the following outline of the paper. After introducing the syntax and semantics of SPARQL in Section 2, we formally define our new fragment in Section 3. In Section 4, we show that, similarly to the well-designed case, weakly well-designed queries can be transformed to an intuitive normal form, which allows for a natural graphical representation as *constraint pattern trees*. Using this representation, in Section 5, we formally show that the step from well-designed to weakly well-designed queries does not increase complexity of query evaluation; minimal relaxations of weak well-designedness, however, already lead to a complexity jump. In Section 6, we compare the expressive power of our fragment (and its extensions with additional operators) with well-designed queries and unrestricted SPARQL queries; in particular, we show that the expressivity of weakly well-designed queries lies strictly in-between well-designed and unrestricted queries. In Section 7, we study static analysis problems for weakly well-designed queries and establish Π_2^p-completeness of equivalence, containment, and subsumption. Finally, in Section 8, we detail our analysis of DBpedia logs.

2 SPARQL Query Language

We begin by formally introducing the syntax and semantics of SPARQL that we adopt in this paper. Our formal setup mostly follows [31], which has some differences from the W3C specification [35, 20]; in particular, we use two-placed OPT and two-valued FILTER (conditional OPT and errors in FILTER evaluation as in the standard are expressible in our formalisation [5]), and adopt set semantics, leaving multiset answers for future work.

RDF Graphs. An RDF graph is a labeled graph where nodes can also serve as edge labels. Formally, let $\mathbf{I}$ be a set of *IRIs*. Then an *RDF triple* is a tuple (s, p, o) from $\mathbf{I} \times \mathbf{I} \times \mathbf{I}$, where s is called *subject*, p *predicate*, and o *object*. An *RDF graph* is a finite set of RDF triples.

SPARQL Syntax. Let $\mathbf{X}$ be an infinite set $\{?x, ?y, \ldots\}$ of *variables*, disjoint from $\mathbf{I}$. *Filter constraints* are conditions of the form

- $\top$, $?x = u$, $?x = ?y$, or $\mathit{bound}(?x)$ for $?x, ?y$ in $\mathbf{X}$ and $u \in \mathbf{I}$ (*atomic constraints*),
- $\neg R_1$, $R_1 \wedge R_2$, or $R_1 \vee R_2$ for filter constraints R_1 and R_2.

A *basic pattern* is a set of triples from $(\mathbf{I} \cup \mathbf{X}) \times (\mathbf{I} \cup \mathbf{X}) \times (\mathbf{I} \cup \mathbf{X})$. Then, SPARQL *(graph) patterns* P are defined by the grammar

$$P ::= B \mid (P \,\mathsf{AND}\, P) \mid (P \,\mathsf{OPT}\, P) \mid (P \,\mathsf{UNION}\, P) \mid (P \,\mathsf{FILTER}\, R),$$

where B ranges over basic patterns and R over filter constraints. Additionally, we require all filter constraints to be *safe*, that is, $\mathsf{vars}(R) \subseteq \mathsf{vars}(P)$ for every pattern $(P \,\mathsf{FILTER}\, R)$, where $\mathsf{vars}(S)$ is the set of all variables in S (which can be pattern, constraint, etc.) When

needed, we distinguish between patterns by their top-level operator (e.g., OPT-pattern or FILTER-pattern). The set of all triples in basic patterns of a pattern P is denoted $\mathsf{triples}(P)$.

We write $\mathcal{U}$ for the set of all patterns. We also distinguish the fragment $\mathcal{P}$ of $\mathcal{U}$ that consists of all UNION-*free* patterns, i.e., patterns that do not use the UNION operator.

Projection is realised in SPARQL by means of *queries with select result form*, or *queries* for short, which are expressions of the form

$$\mathsf{SELECT}\ X\ \mathsf{WHERE}\ P, \tag{4}$$

where X is a set of variables and P is a graph pattern. We write $\mathcal{S}$ for the set of all queries.

Note that every pattern P can be seen as a query of the form (4) where $X = \mathsf{vars}(P)$. Hence, all definitions that refer to "queries" implicitly extend to patterns in the obvious way.

SPARQL Semantics. The semantics of graph patterns is defined in terms of *mappings*, that is, partial functions from variables to IRIs. The *domain* $\mathsf{dom}(\mu)$ of a mapping μ is the set of variables on which μ is defined. Two mappings μ_1 and μ_2 are *compatible* (written $\mu_1 \sim \mu_2$) if $\mu_1(?x) = \mu_2(?x)$ for all variables $?x \in \mathsf{dom}(\mu_1) \cap \mathsf{dom}(\mu_2)$. If $\mu_1 \sim \mu_2$, then $\mu_1 \cup \mu_2$ constitutes a mapping that coincides with μ_1 on $\mathsf{dom}(\mu_1)$ and with μ_2 on $\mathsf{dom}(\mu_2)$. Given two sets of mappings Ω_1 and Ω_2, we define their *join*, *union* and *difference* as follows:

$$\begin{array}{rcl} \Omega_1 \bowtie \Omega_2 & = & \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \text{ and } \mu_1 \sim \mu_2\}, \\ \Omega_1 \cup \Omega_2 & = & \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}, \\ \Omega_1 \setminus \Omega_2 & = & \{\mu_1 \mid \mu_1 \in \Omega_1, \mu_1 \not\sim \mu_2 \text{ for all } \mu_2 \in \Omega_2\}. \end{array}$$

Based on these, the *left outer join* operation is defined as $\Omega_1 \mathbin{⟕} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$. Given a graph G, the *evaluation* $[\![P]\!]_G$ of a graph pattern P over G is defined as follows:

1. if B is a basic pattern, then $[\![B]\!]_G = \{\mu : \mathsf{vars}(B) \to \mathbf{I} \mid \mu(B) \subseteq G\}$;
2. $[\![(P_1\ \mathsf{AND}\ P_2)]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$;
3. $[\![(P_1\ \mathsf{OPT}\ P_2)]\!]_G = [\![P_1]\!]_G \mathbin{⟕} [\![P_2]\!]_G$;
4. $[\![(P_1\ \mathsf{UNION}\ P_2)]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$;
5. $[\![(P'\ \mathsf{FILTER}\ R)]\!]_G = \{\mu \mid \mu \in [\![P']\!]_G \text{ and } \mu \models R\}$,
   where μ *satisfies* a filter constraint R, denoted by $\mu \models R$, if one of the following holds:
   - R is $\top$;
   - R is $?x = u$, $?x \in \mathsf{dom}(\mu)$, and $\mu(?x) = u$;
   - R is $?x = ?y$, $\{?x, ?y\} \subseteq \mathsf{dom}(\mu)$, and $\mu(?x) = \mu(?y)$;
   - R is $\mathit{bound}(?x)$ and $?x \in \mathsf{dom}(\mu)$;
   - R is a Boolean combination of filter constraints evaluating to *true* under the usual interpretation of $\neg, \wedge$, and $\vee$.

Let $\mu|_X$ be the *projection* of a mapping μ to variables X, that is, $\mu|_X(?x) = \mu(?x)$ if $?x \in X$ and $\mu|_X(?x)$ is undefined if $?x \notin X$. The *evaluation* $[\![Q]\!]_G$ of a query Q of the form (4) is the set of all mappings $\mu|_X$ such that $\mu \in [\![P]\!]_G$.

Finally, a *solution* to a query (or pattern) Q over G is a mapping μ such that $\mu \in [\![Q]\!]_G$.

3 Weakly Well-Designed Patterns

We begin by recalling the notion of well-designed patterns and then formulate our generalisation. For now, we focus on the AND-OPT-FILTER fragment $\mathcal{P}$, leaving the operators UNION and SELECT for later sections.

Note that a given pattern can occur more than once within a larger pattern. In what follows we will sometimes need to distinguish between a (sub-)pattern P as a possibly repeated building block of another pattern P' and its *occurrences* in P', that is, unique subtrees in the parse tree. Then, the *left (right) argument* of an occurrence i is the subtree rooted in the left (right) child of the root of i in the parse tree, and an occurrence i is *inside* an occurrence j if the root of i is a successor of the root of j.

▶ **Definition 1** (Pérez et al. [31]). A pattern P from $\mathcal{P}$ is *well-designed* (or *wd-pattern*, for short) if for every occurrence i of an OPT-pattern P_1 OPT P_2 in P the variables from $\mathsf{vars}(P_2) \setminus \mathsf{vars}(P_1)$ occur in P only inside (the labels of) i.

We write $\mathcal{P}_{\text{wd}}$ for the fragment of wd-patterns. Such patterns comply with the basic intuition for optional matching in SPARQL: *"do not reject (solutions) because some part of the query pattern does not match"* [20]; indeed, our canonical use case (1) is clearly well-designed. Evaluation of wd-patterns, that is, checking if $\mu \in [\![P]\!]_G$ for a mapping μ, graph G and pattern $P \in \mathcal{P}_{\text{wd}}$, is coNP-complete, as opposed to PSPACE-complete for $\mathcal{P}$ [31, 36]. The high complexity of unrestricted patterns is partially due to the fact that unrestricted combinations of OPT and FILTER allow to express nesting of the difference operator MINUS with semantics $[\![P_1 \text{ MINUS } P_2]\!]_G = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$ (for non-empty P_1 and P_2):

$$P_1 \text{ MINUS } P_2 \equiv (P_1 \text{ OPT } (P_2 \text{ AND } (?x, ?y, ?z))) \text{ FILTER } \neg \mathit{bound}(?x). \tag{5}$$

This property is well-known [5, 31], and has been usually considered the main source of non-well-designed patterns in practice. We challenge this claim by answering differently the question on the prevalent structure of real-life queries beyond the well-designed fragment. This question is not just of theoretical interest: as previous studies [32] show (and our analysis confirms), about half of queries with OPT asked over DBpedia are not well-designed.

Next we discuss two sources of non-well-designedness in patterns as revealed by the example queries (2) and (3) in the introduction – one based on OPT and another on FILTER.

Source 1. There are two substantially different ways of nesting the OPT operator in patterns:

$$P_1 \text{ OPT } (P_2 \text{ OPT } P_3), \qquad (\texttt{Opt-R}) \qquad\qquad (P_1 \text{ OPT } P_2) \text{ OPT } P_3. \qquad (\texttt{Opt-L})$$

Non-well-designed nesting of type (`Opt-R`) is responsible for the PSPACE-hardness of query evaluation [31, 36]. Moreover, such nesting is not very intuitive. On the contrary, as we saw in the introduction, non-well-designed nesting of type (`Opt-L`) can be used for prioritising some parts of patterns to others, and is indeed used in real life. As we will see later, nesting of type (`Opt-L`) cannot lead to high complexity of evaluation.

Source 2. Well-designedness can be violated by using "dangerous" variables from the right side of OPT in filter constraints. In particular, patterns of the form $(P_1 \text{ OPT } P_2) \text{ FILTER } R$ with R using a variable from $\mathsf{vars}(P_2) \setminus \mathsf{vars}(P_1)$ are not well-designed, but rather frequent in practice. However, such patterns almost never occur inside the right argument of other OPT-patterns. We will see that if we restrict the usage of such filters to the "top level", we preserve the good computational properties of wd-patterns.

Motivated by these observations, we considerably generalise the notion of wd-patterns to allow for useful queries like (2) and (3) while retaining important properties of such patterns.

We start with two auxiliary notions. Given a pattern P, an occurrence i_1 in P *dominates* another occurrence i_2 if there exists an occurrence j of an OPT-pattern such that i_1 is inside the left argument of j and i_2 is inside the right argument. An occurrence i of a FILTER-pattern P' FILTER R in P is *top-level* if there is no occurrence j of an OPT-pattern such that i is inside the right argument of j.

▶ **Definition 2.** A pattern $P \in \mathcal{P}$ is *weakly well-designed* (*wwd-pattern*) if, for each occurrence i of an OPT-subpattern P_1 OPT P_2, the variables in $\mathsf{vars}(P_2) \setminus \mathsf{vars}(P_1)$ appear outside i only in

- subpatterns whose occurrences are dominated by i, and
- constraints of top-level occurrences of FILTER-patterns.

We write $\mathcal{P}_{\mathrm{wwd}}$ for the fragment of wwd-patterns. They extend wd-patterns by allowing variables from the right argument of an OPT-subpattern that are not "guarded" by the left argument to appear in certain positions outside of the subpattern. Note that the patterns of queries (4) and (3) are wwd-patterns. Also, patterns which allow only for OPT nesting of type (`Opt-L`) are always weakly well-designed, same as the pattern in the right hand side of (5), which expresses MINUS. However, patterns that have subpatterns of the latter form in the right argument of OPT are not weakly well-designed. Next we give a few more examples.

▶ **Example 3.** Consider the following patterns:

$$((?x, a, a) \text{ OPT } ((?x, b, ?y) \text{ OPT } (?y, c, ?z))) \text{ OPT } (?x, d, ?z), \tag{6}$$

$$((?x, a, a) \text{ OPT } (?x, d, ?z)) \text{ OPT } ((?x, b, ?y) \text{ OPT } (?y, c, ?z)), \tag{7}$$

$$(((?u, f, ?v) \text{ OPT } (?u, g, ?w)) \text{ FILTER } ?v \neq ?w) \text{ OPT } (?u, h, ?s), \tag{8}$$

$$(?u, h, ?s) \text{ OPT } (((?u, f, ?v) \text{ OPT } (?u, g, ?w)) \text{ FILTER } ?v \neq ?w). \tag{9}$$

Pattern (6) is not well-designed because of variable $?z$, but is weakly well-designed since the occurrence of $(?y, c, ?z)$ dominates $(?x, d, ?z)$. However, the similar pattern (7) is not weakly well-designed because the occurrence of the inner OPT-pattern with the second occurrence of $?z$ does not dominate the first. Pattern (8) is weakly well-designed since the FILTER-pattern is top-level (we write $?x \neq ?y$ for $\neg(?x = ?y)$), but pattern (9) is not, because of variable $?w$ in a non-top-level FILTER.

▶ **Proposition 4.** *Checking whether a pattern P belongs to the fragment $\mathcal{P}_{\mathrm{wwd}}$ can be done in time $O(|P|^2)$, where $|P|$ is the length of the string representation of P.*

4 OPT-FILTER-Normal Form and Constraint Pattern Trees

One of the key properties of wd-patterns is that they can always be converted to a so-called OPT-normal form, in which all AND- and FILTER-subpatterns are OPT-free [31]. Also, FILTER-free patterns in OPT-normal form can be naturally represented as trees [27, 33], which gives a good intuition for the evaluation and optimisation of such patterns. In this section, we show that these notions can be generalised to wwd-patterns.

▶ **Definition 5.** A pattern $P \in \mathcal{P}$ is in OPT-FILTER-*normal form* (or OF-*normal form* for short) if it adheres to the grammar

$$P ::= F \mid (P \text{ FILTER } R) \mid (P \text{ OPT } S), \qquad S ::= F \mid (S \text{ OPT } S), \qquad F ::= (B \text{ FILTER } R),$$

where B ranges over basic patterns and R over filter constraints.

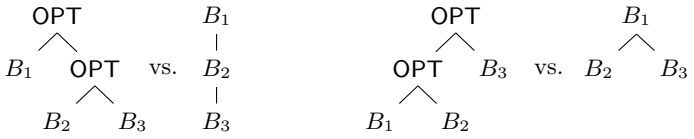


Figure 2 Parse trees vs. constraint pattern trees for patterns **(a)** B_1 OPT $(B_2$ OPT $B_3)$ and **(b)** $(B_1$ OPT $B_2)$ OPT B_3, with B_1, B_2, and B_3 basic patterns.

In other words, the parse tree of a pattern in OF-normal form can be stratified as follows:

1. (occurrences of) basic patterns as the bottom layer,
2. a FILTER on top of each basic pattern as the middle layer,
3. a combination of OPT and FILTER as the top layer;

moreover, each occurrence of a FILTER-pattern in the top layer is top-level. Note that our normal form is AND-free: all conjunctions are expressed via basic patterns.

▶ **Example 6.** None of the four patterns in Example 3 are in OF-normal form. However, the first three of them can be easily normalised by replacing each triple t with $t^\top$, where $P^\top$ is an abbreviation of P FILTER $\top$ for a pattern P. Also, compare the pattern

$$(((?x, a, a)^\top \text{ OPT } (?x, b, ?y)^\top) \text{ OPT } ((?x, b, ?z)^\top \text{ OPT } (?z, c, ?u)^\top)) \text{ FILTER } ?u \neq ?x, \quad (10)$$

which is in OF-normal form, with the very similar pattern

$$(((?x, a, a)^\top \text{ OPT } (?x, b, ?u)^\top) \text{ OPT } ((?x, b, ?z)^\top \text{ OPT } (?z, c, ?u)^\top) \text{ FILTER } ?u \neq ?z),$$

which is not, because the outer FILTER is in the right argument of the outermost OPT.

As shown by Letelier et al. [27], FILTER-free patterns in OPT-normal form can be represented by means of so-called pattern trees. We next show that this representation can be naturally extended to patterns in OF-normal form.

Let P be a pattern in OF-normal form. The *constraint pattern tree (CPT)* $\mathcal{T}(P)$ of P is the directed ordered labelled rooted tree recursively constructed as follows (in this definition we abuse notation and confuse patterns and their occurrences; strictly speaking, we create a fresh sub-tree for each occurrence, so the resulting object is indeed always a tree):

1. if B is a basic pattern then $\mathcal{T}(B \text{ FILTER } R)$ is a single node v labelled by the pair (B, R);
2. if P' is not a basic pattern then $\mathcal{T}(P' \text{ FILTER } R)$ is obtained by adding a *special* node labelled by R as the last child of the root of $\mathcal{T}(P')$;
3. $\mathcal{T}(P_1 \text{ OPT } P_2)$ is the tree obtained from $\mathcal{T}(P_1)$ and $\mathcal{T}(P_2)$ by adding the root of $\mathcal{T}(P_2)$ as the last child of the root of $\mathcal{T}(P_1)$.

By definition, there is a one-to-one correspondence between patterns in OF-normal form and CPTs. Hence, such trees can be seen as a convenient representation of patterns in OF-normal form.

Unlike parse trees, which represent the syntactic shape of patterns, CPTs show the semantic structure of OPT and FILTER nesting. Figure 2 shows how OPT nestings of types (`Opt-R`) and (`Opt-L`) are represented in both formats. Note that CPTs treat different FILTER-subpatterns differently: if the filter is over a basic pattern, the constraint of the FILTER is paired with this pattern; however, if the filter is over an OPT-subpattern, then the constraint is represented by a separate special node. Moreover, since in the second case

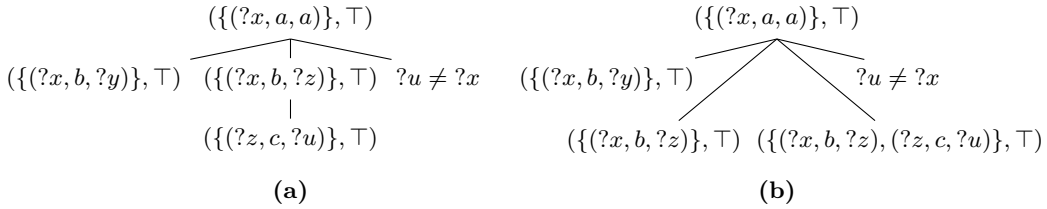


Figure 3 Constraint pattern trees of **(a)** $(((?x, a, a)^{\top}\ \mathsf{OPT}\ (?x, b, ?y)^{\top})\ \mathsf{OPT}\ ((?x, b, ?z)^{\top}\ \mathsf{OPT}\ (?z, c, ?u)^{\top}))\ \mathsf{FILTER}\ ?u \neq ?x$ (i.e., pattern (10)) and **(b)** equivalent pattern in "flat" form (13).

the FILTER-pattern must be top-level, special nodes can only occur in CPTs as children of the root. For instance, the CPT of the example pattern (10) is given in Figure 3(a).

Each wwd-pattern can be converted to OF-normal form and hence can be represented by a CPT. To prove this statement we make use of a number of equivalences. Formally, a pattern P_1 is *equivalent* to a pattern P_2 (written $P_1 \equiv P_2$) if $[\![P_1]\!]_G = [\![P_2]\!]_G$ holds for any graph G. There are several equivalences, such as associativity and commutativity of AND, as well as filter decompositions, such as $P\ \mathsf{FILTER}\ (R_1 \wedge R_2) \equiv (P\ \mathsf{FILTER}\ R_1)\ \mathsf{FILTER}\ R_2$, which hold for all patterns (see [36] for an extensive list). Moreover, the key equivalences used in [31] for normalising wd-patterns can easily be adapted to serve our needs.

▶ **Proposition 7.** *Let P_1, P_2, P_3 be patterns and R a filter constraint such that $\mathsf{vars}(P_2) \cap \mathsf{vars}(P_3) \subseteq \mathsf{vars}(P_1)$ and $\mathsf{vars}(P_2) \cap \mathsf{vars}(R) \subseteq \mathsf{vars}(P_1)$. Then the following equivalences hold:*

$$(P_1\ \mathsf{OPT}\ P_2)\ \mathsf{AND}\ P_3 \equiv (P_1\ \mathsf{AND}\ P_3)\ \mathsf{OPT}\ P_2,$$
$$(P_1\ \mathsf{OPT}\ P_2)\ \mathsf{FILTER}\ R \equiv (P_1\ \mathsf{FILTER}\ R)\ \mathsf{OPT}\ P_2.$$

Since all the equivalences preserve weak well-designedness, we obtain the desired result.

▶ **Proposition 8.** *Each wwd-pattern P is equivalent to a wwd-pattern in OF-normal form of size $O(|P|)$.*

Relying on this proposition, in the rest of the paper we silently assume that all wwd-patterns are in OF-normal form and hence can be represented by CPTs.

We next transfer the notion of weak well-designedness to CPTs. Let $\prec$ be the strict topological sorting of the nodes in $\mathcal{T}(P)$, computed by a depth first search traversal visiting the children of a node according to their ordering (i.e., $v \prec u$ holds if v is visited before u).

▶ **Proposition 9.** *A pattern P in OF-normal form is weakly well-designed if and only if, for each edge (v, u) in its CPT $\mathcal{T}(P)$, every variable $?x \in \mathsf{vars}(u) \setminus \mathsf{vars}(v)$ occurs only in nodes w such that $v \prec w$. The pattern is well-designed if and only if for every variable $?x$ in P the set of all nodes v in $\mathcal{T}(P)$ with $?x \in \mathsf{vars}(v)$ is connected.*

Note that if a pattern is FILTER-free, its OF-normal form coincides with the OPT-normal form in [31] (modulo tautological filters), and its CPT is the pattern tree from [27, 33]. In fact, the second part of Proposition 9 generalises an observation from [27] to the case with filters. An important difference to pattern trees is that in our case the order of children of a node is semantically relevant since wwd-patterns do not satisfy the equivalence

$$(P_1\ \mathsf{OPT}\ P_2)\ \mathsf{OPT}\ P_3 \equiv (P_1\ \mathsf{OPT}\ P_3)\ \mathsf{OPT}\ P_2. \tag{11}$$

This equivalence, established in [30], holds whenever $(\mathsf{vars}(P_2) \cap \mathsf{vars}(P_3)) \subseteq \mathsf{vars}(P_1)$, which is always the case for wd-patterns but not for wwd-patterns, as can be seen on query (2).

We conclude this section with a property that is unique to wwd-patterns: each wwd-pattern is equivalent to a pattern whose corresponding CPT has depth one.

▶ **Definition 10.** A pattern in $\mathcal{P}$ is in *depth-one normal form* if it has the structure

$$(\cdots((B\ \mathsf{op}_1\ S_1)\ \mathsf{op}_2\ S_2)\cdots)\ \mathsf{op}_n\ S_n, \tag{12}$$

where B is a basic pattern and each $\mathsf{op}_i\ S_i$, $1 \leq i \leq n$, is either $\mathsf{OPT}\,(B_i\ \mathsf{FILTER}\ R_i)$ with B_i a basic pattern and R_i a filter constraint, or just $\mathsf{FILTER}\,R_i$.

To show that each wwd-pattern can be brought to this form we use another equivalence.

▶ **Proposition 11.** *For patterns P_1, P_2, P_3 with $\mathsf{vars}(P_1) \cap \mathsf{vars}(P_3) \subseteq \mathsf{vars}(P_2)$ it holds that*

$$P_1\ \mathsf{OPT}\ (P_2\ \mathsf{OPT}\ P_3) \equiv (P_1\ \mathsf{OPT}\ P_2)\ \mathsf{OPT}\ (P_2\ \mathsf{AND}\ P_3). \tag{13}$$

Applied from left to right, equivalence (13) preserves weak well-designedness (but not well-designedness). Each such application transforms a weakly well-designed OPT nesting of type (`Opt-R`) to a nesting of type (`Opt-L`), decreasing the depth of the CPT.

▶ **Corollary 12.** *Every wwd-pattern is equivalent to a wwd-pattern in depth-one normal form.*

For instance, pattern (10) is equivalent to the pattern

$$((((?x, a, a)^{\top}\mathsf{OPT}(?x, b, ?y)^{\top})\mathsf{OPT}(?x, b, ?z)^{\top})\mathsf{OPT}\{(?x, b, ?z), (?z, c, ?u)\}^{\top})\mathsf{FILTER}\ ?u \neq ?x,$$

represented by the CPT in Figure 3(b). Such "flat" patterns are attractive in practice because of their regular structure. However, "flattening" a pattern can incur an exponential blowup in size. Hence, in the rest of the paper we consider arbitrary wwd-patterns in OF-normal form rather than restricting our attention to depth-one-normal patterns.

5 Evaluation of wwd-Patterns

In this section, we look at the query answering problem for wwd-patterns and their extensions with union and projection. We show that in all three cases, complexity remains the same as for wd-patterns. To obtain these results, we develop several new techniques.

Formally, we look at the following decision problem for a given SPARQL fragment $\mathcal{L}$.

Eval($\mathcal{L}$)	**Input:**	Graph G, query $Q \in \mathcal{L}$, and mapping μ
	Question:	Does μ belong to $[\![Q]\!]_G$?

It is known that Eval($\mathcal{U}$) for general patterns $\mathcal{U}$ is PSPACE-complete [31], and the result easily propagates to queries with projection (i.e., $\mathcal{S}$) [27]. For wd-patterns, the evaluation problem is coNP-complete, and can be solved by exploiting the following idea [27].

Suppose we are given a wd-pattern P in OPT-normal form (for simplicity, suppose P is FILTER-free), a graph G, and a mapping μ. First, we look for a subtree of $\mathcal{T}(P)$ that includes the root of $\mathcal{T}(P)$, contains precisely the variables in $\mathsf{dom}(\mu)$, and "matches" G under μ (i.e., images of all its triples under μ are contained in G). This is doable in polynomial time. If such a subtree does not exist, then μ cannot be a solution. Otherwise, the subtree witnesses that μ is a part of a solution to P. Finally, to verify that μ is a complete solution, we need to check that the subtree is maximal, that is, cannot be extended to any more nodes in $\mathcal{T}(P)$ with a match in G. There are linearly many such nodes to check, and each check can be performed in coNP. So, the overall algorithm runs in coNP.

Inspired by this idea, we next show that the low evaluation complexity of wd-patterns transfers to wwd-patterns by developing a coNP algorithm for $\textsc{Eval}(\mathcal{P}_{\text{wwd}})$.

Let P be a wwd-pattern in OF-normal form. An *r-subtree* of $\mathcal{T}(P)$ is a subtree containing the root of $\mathcal{T}(P)$ and all its special children. Every r-subtree is also a CPT representing a wwd-pattern that can be obtained from P by dropping the right arguments of some OPT-subpatterns (i.e., a pattern P' with $P' \trianglelefteq P$ in the notation of [31]). A *child* of an r-subtree $\mathcal{T}(P')$ of $\mathcal{T}(P)$ is a node in $\mathcal{T}(P)$ that is not contained in $\mathcal{T}(P')$ but whose parent is.

▶ **Definition 13.** A mapping μ is a *potential partial solution* (or *pp-solution* for short) to a wwd-pattern P over a graph G if there is an r-subtree $\mathcal{T}(P')$ of $\mathcal{T}(P)$ such that $\mathsf{dom}(\mu) = \mathsf{vars}(P')$, $\mu(\mathsf{triples}(P')) \subseteq G$, and $\mu \models R$ for the constraint R of any ordinary node in $\mathcal{T}(P')$.

A pp-solution μ to P over G can be witnessed by several r-subtrees. However, the union of such r-subtrees is also a witness. Hence, there exists a unique maximal witnessing r-subtree, denoted $\mathcal{T}(P_\mu)$, with P_μ being the corresponding wwd-pattern.

Potential partial solutions generalise "partial solutions" as defined in [31] for wd-patterns. There, every "partial solution" is either a solution or can be extended to one. This is not the case for wwd-patterns. While every solution is clearly a pp-solution, not every pp-solution can be extended to a real one. Real solutions may not just extend pp-solutions by assigning previously undefined variables but can also override variable bindings established in some node v of $\mathcal{T}(P_\mu)$ by extending $\mathcal{T}(P_\mu)$ to a child that precedes v according to the order $\prec$.

An additional complication is the presence of non-well-designed top-level filters. Note that pp-solutions are only required to satisfy the constraints of ordinary nodes in the corresponding CPT, thus ignoring top-level filters. Indeed, requiring pp-solutions to satisfy constraints of top-level filters would be too strong since real solutions do not generally satisfy this property, as demonstrated by the following example.

▶ **Example 14.** Consider the graph $G = \{(1, a, 1), (3, a, 3)\}$ and wwd-pattern

$$P = (((?x, a, 1)\ \mathsf{OPT}\ (?y, a, 2))\ \mathsf{FILTER}\ \neg \mathit{bound}(?y))\ \mathsf{OPT}\ (?y, a, 3).$$

The mapping $\mu = \{?x \mapsto 1, ?y \mapsto 3\}$ is a solution to P over G, but $\mu \not\models \neg \mathit{bound}(?y)$.

We now present a characterisation of solutions for wwd-patterns in terms of pp-solutions that (a)takes into account that not every pp-solution can be extended to a real solution and (b) ensures correct treatment of non-well-designed top-level filters. For this we need some more notation. Given a wwd-pattern P, a node v in $\mathcal{T}(P)$, a graph G, and a pp-solution μ to P over G, let $\mu|_v$ be the projection $\mu|_X$ of μ to the set X of all variables appearing in nodes u of $\mathcal{T}(P_\mu)$ such that $u \prec v$. A mapping μ_1 is *subsumed* by a mapping μ_2 (written $\mu_1 \sqsubseteq \mu_2$) if $\mu_1 \sim \mu_2$ and $\mathsf{dom}(\mu_1) \subseteq \mathsf{dom}(\mu_2)$ (this notion is from [31, 8]).

▶ **Lemma 15.** *A mapping μ is a solution to a wwd-pattern P over a graph G if and only if*

1. *μ is a pp-solution to P over G;*
2. *for any child v of $\mathcal{T}(P_\mu)$ labelled with (B, R) there is no μ' such that $\mu|_v \sqsubseteq \mu'$, $\mu' \models R$, and $\mu'(B) \subseteq G$;*
3. *$\mu|_s \models R$ for any special node s in $\mathcal{T}(P)$ labelled with R.*

Intuitively, a pp-solution μ needs to satisfy two conditions to be a real solution to a wwd-pattern P. First, $\mu|_v$ (as opposed to μ for wd-patterns) must be non-extendable to v for any child v of $\mathcal{T}(P_\mu)$. Indeed, if such an extension exists, then it is either possible to provide

bindings for some variables that are undefined in μ, or some variables from $\mathsf{dom}(\mu)$ can be assigned different values of higher "priority" than the corresponding values in μ. Second, every top-level filter R labelling a node s needs to be satisfied by $\mu|_s$, which is precisely the part of μ bound by the subpattern of P that is paired with R in the FILTER-pattern.

Checking whether a mapping μ satisfies this characterisation is feasible in coNP: testing whether μ is a pp-solution takes polynomial time, same as computing the maximal witnessing tree $\mathcal{T}(P_\mu)$; to check that (the relevant part of) $\mathcal{T}(P_\mu)$ is not extendable to any of its children we need to consider linearly many children, and each check is in coNP; finally, the checks for top-level filters are again polynomial. Hence, we obtain the following theorem, where the hardness part follows from the coNP-hardness for wd-patterns [31].

▶ **Theorem 16.** $\textsc{Eval}(\mathcal{P}_{\text{wwd}})$ *is* coNP*-complete.*

Pérez et al. [31] extended wd-patterns to UNION by considering *unions of wd-patterns*, that is, patterns of the form P_1 UNION ... UNION P_n with all $P_i \in \mathcal{P}_{\text{wd}}$. We denote the resulting fragment by $\mathcal{U}_{\text{wd}}$. This syntactic restriction on the use of UNION in $\mathcal{U}_{\text{wd}}$ is motivated by the fact that any pattern in $\mathcal{U}$ can be equivalently expressed as a union of UNION-free patterns [31]. We denote the fragment of all queries over patterns in $\mathcal{U}_{\text{wd}}$ as $\mathcal{S}_{\text{wd}}$. Similarly, we write $\mathcal{U}_{\text{wwd}}$ for unions of wwd-patterns and $\mathcal{S}_{\text{wwd}}$ for queries over unions of wwd-patterns.

Analogously to the well-designed case, Theorem 16 extends to fragments $\mathcal{U}_{\text{wwd}}$ and $\mathcal{S}_{\text{wwd}}$.

▶ **Corollary 17.** $\textsc{Eval}(\mathcal{U}_{\text{wwd}})$ *is* coNP*-complete and* $\textsc{Eval}(\mathcal{S}_{\text{wwd}})$ *is* Σ_2^p*-complete.*

The coNP-algorithm for $\mathcal{U}_{\text{wwd}}$ is obtained simply by applying the algorithm for $\mathcal{P}_{\text{wwd}}$ to each pattern in the union. Hardness for $\mathcal{S}_{\text{wwd}}$ follows from the hardness of the well-designed case [27], while for membership we just guess the values of the existential variables and then call a coNP-oracle for $\mathcal{U}_{\text{wwd}}$ on the resulting mapping and the normalised body of the query.

Hence, the complexity of evaluation for wwd-patterns is the same as for wd-patterns. We next show that wwd-patterns are, in a certain sense, a maximal extension of wd-patterns that preserves coNP evaluation complexity (under the usual complexity-theoretic assumptions).

There are two possible minimal relaxations of weak well-designedness that allow for basic patterns and filter constraints of arbitrary shape. We show that both lead to Π_2^p-hardness.

The first such relaxation is to allow for at least some non-well-designed OPT-nesting of type (`Opt-R`). However, even a minimal extension of this sort increases complexity. To see this, consider the fragment $\mathcal{P}_{\text{opt-r}}$ of patterns of the form B_1 OPT $(B_2$ OPT $B_3)$, where B_1, B_2 and B_3 are basic patterns. Intuitively, $\mathcal{P}_{\text{opt-r}}$ allows for the most simple form of non-well-designed nesting of type (`Opt-R`).

The other syntactic relaxation is to allow for some non-well-designed non-top-level filters. However, while requiring special nodes to be children of the root may look somewhat ad-hoc, it cannot be substantially relaxed. Consider the fragment $\mathcal{P}_{\text{filter-2}}$ of patterns of the form B_1 OPT $((B_2$ OPT $B_3)$ FILTER $R)$, where B_1, B_2 and B_3 are basic patterns such that $\mathsf{vars}(B_3) \cap \mathsf{vars}(B_1) \subseteq \mathsf{vars}(B_2)$, and R is a filter constraint. Intuitively, $\mathcal{P}_{\text{filter-2}}$ allows for the simplest form of "second-level" filters.

▶ **Proposition 18.** *The problems* $\textsc{Eval}(\mathcal{P}_{\text{opt-r}})$ *and* $\textsc{Eval}(\mathcal{P}_{\text{filter-2}})$ *are* Π_2^p*-complete.*

Proposition 18 implies that $\mathcal{P}_{\text{wwd}}$ is a maximal fragment of $\mathcal{P}$ that does not impose structural restrictions on basic patterns or filter constraints and has a coNP evaluation algorithm (assuming coNP $\neq \Pi_2^p$). Hence, going beyond wwd-patterns while preserving good computational properties requires more refined restrictions, as done, for example, in [27, Section 4].

6 Expressivity of wwd-Patterns and their Extensions

In this section, we analyse the expressive power of our fragments. Formally, a language $\mathcal{L}_1$ has the *same expressive power* as a language $\mathcal{L}_2$ (written $\mathcal{L}_1 \sim \mathcal{L}_2$) if for every query Q_2 in $\mathcal{L}_2$ there is a query Q_1 in $\mathcal{L}_1$ such that $Q_2 \equiv Q_1$ and vice versa; $\mathcal{L}_1$ is *strictly more expressive* than $\mathcal{L}_2$ (written $\mathcal{L}_2 < \mathcal{L}_1$) if the property holds in the forward but not in the backward direction. We begin by establishing $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}_{\mathrm{wwd}} < \mathcal{P}$. Then we proceed to unions, showing that $\mathcal{U}_{\mathrm{wd}} < \mathcal{U}_{\mathrm{wwd}} < \mathcal{U}$. Finally, we establish $\mathcal{S}_{\mathrm{wwd}} \sim \mathcal{S}$, i.e., wwd-patterns with union and projection have the full expressive power of SPARQL (whereas it is known that $\mathcal{S}_{\mathrm{wd}} < \mathcal{S}$ [31], which then implies $\mathcal{S}_{\mathrm{wd}} < \mathcal{S}_{\mathrm{wwd}}$).

Following [31, 8], a set of mappings Ω_1 is *subsumed* by a set of mappings Ω_2 (written $\Omega_1 \sqsubseteq \Omega_2$) if for every $\mu_1 \in \Omega_1$ there exists a mapping $\mu_2 \in \Omega_2$ such that $\mu_1 \sqsubseteq \mu_2$. A query Q is *weakly monotone* if $[\![Q]\!]_{G_1} \sqsubseteq [\![Q]\!]_{G_2}$ for any two graphs G_1 and G_2 with $G_1 \subseteq G_2$, and a fragment $\mathcal{L}$ is *weakly monotone* if it contains only weakly monotone queries. Arenas and Pérez [8] showed that, unlike $\mathcal{P}$, the fragment $\mathcal{P}_{\mathrm{wd}}$ is weakly monotone, and hence $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}$.

▶ **Example 19** (Pérez et al. [31])**.** Consider the non-well-designed pattern

$$P = (?x, a, 1)\ \mathsf{OPT}\ ((?y, a, 2)\ \mathsf{OPT}\ (?x, a, 3))$$

as well as graphs $G_1 = \{(1, a, 1), (2, a, 2)\}$ and $G_2 = G_1 \cup \{(3, a, 3)\}$. Then $\mu_1 = \{?x \mapsto 1, ?y \mapsto 2\}$ is the only mapping in $[\![P]\!]_{G_1}$ while $\mu_2 = \{?x \mapsto 1\}$ is the only mapping in $[\![P]\!]_{G_2}$. Hence $[\![P]\!]_{G_1} \not\sqsubseteq [\![P]\!]_{G_2}$, meaning P is not weakly monotone.

Analogously, we show that $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}_{\mathrm{wwd}}$ by observing that $\mathcal{P}_{\mathrm{wwd}}$ is not weakly monotone. Indeed, the pattern in example query (2) violates weak monotonicity: if a graph G contains the triple (`P1`, `v_card:name`, *Anastasia*) but no triple of the form (`P1`, `foaf:name`, u) for any IRI u, then extending G with (`P1`, `foaf:name`, *Ana*), that is, adding more reliable information about the name of `P1`, does not extend the original solution $\{?i \mapsto$ `P1`$, ?n \mapsto$ *Anastasia*$\}$ but modifies it by overriding the value of $?n$. Since $\mathcal{P}_{\mathrm{wd}} \subseteq \mathcal{P}_{\mathrm{wwd}}$, we conclude that $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}_{\mathrm{wwd}}$.

To distinguish $\mathcal{P}_{\mathrm{wwd}}$ from $\mathcal{P}$ we need a different property.

▶ **Definition 20.** A query Q is *non-reducing* if for any two graphs G_1, G_2 such that $G_1 \subseteq G_2$ and any mapping $\mu_1 \in [\![Q]\!]_{G_1}$ there is no $\mu_2 \in [\![Q]\!]_{G_2}$ such that $\mu_2 \sqsubset \mu_1$ (i.e., $\mu_2 \sqsubseteq \mu_1$ and $\mu_2 \neq \mu_1$). A fragment $\mathcal{L}$ is *non-reducing* if it contains only non-reducing queries.

Intuitively, for a non-reducing query extending a graph cannot result in a previously bound answer variable becoming unbound. Weakly monotone queries are non-reducing but not vice versa. Moreover, it is easily seen that wwd-patterns are non-reducing.

This property is not generally satisfied by patterns that are not weakly well-designed. For instance, consider again pattern P, graphs G_1, G_2, and mappings μ_1, μ_2 from Example 19. Pattern P is not non-reducing since $\mu_1 \in [\![P]\!]_{G_1}$ and $\mu_2 \in [\![P]\!]_{G_2}$ but $\mu_2 \sqsubset \mu_1$.

▶ **Theorem 21.** *It holds that* $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}_{\mathrm{wwd}} < \mathcal{P}$.

We next compare $\mathcal{U}_{\mathrm{wwd}}$ to $\mathcal{U}_{\mathrm{wd}}$ and $\mathcal{U}$, and $\mathcal{S}_{\mathrm{wwd}}$ to $\mathcal{S}_{\mathrm{wd}}$ and $\mathcal{S}$ (note that neither UNION nor projection via SELECT can be expressed by means of the other operators [37], so adding either construct makes each fragment strictly more expressive). It is easily seen that $\mathcal{U}_{\mathrm{wd}}$ and $\mathcal{S}_{\mathrm{wd}}$ inherit weak monotonicity from $\mathcal{P}_{\mathrm{wd}}$ [31, 27], and hence $\mathcal{U}_{\mathrm{wd}} < \mathcal{U}_{\mathrm{wwd}}$ and $\mathcal{S}_{\mathrm{wd}} < \mathcal{S}_{\mathrm{wwd}}$. Non-reducibility, however, propagates neither to unions nor to projection.

▶ **Example 22.** Consider the following $\mathcal{U}_{\text{wd}}$-pattern with G_1, G_2 and μ_1, μ_2 from Example 19:

$$P = ((?x, a, 1)\ \mathsf{OPT}\ (?y, a, 2))\ \mathsf{UNION}\ (?x, a, 1).$$

We have $\mu_1 \in [\![P]\!]_{G_1}$ and $\mu_2 \in [\![P]\!]_{G_2}$ but $\mu_2 \sqsubset \mu_1$, which is due to the fact that μ_2 is already contained in $[\![P]\!]_{G_1}$ along with μ_1. This is only possible in the presence of UNION since all mappings in the evaluation of a UNION-free pattern are mutually non-subsuming [31].

Thus, to account for UNION, we introduce the following, more delicate property.

▶ **Definition 23.** A query Q is *extension-witnessing* (*e-witnessing*) if for any two graphs $G_1 \subseteq G_2$ and mapping $\mu \in [\![Q]\!]_{G_2}$ such that $\mu \notin [\![Q]\!]_{G_1}$ there is a triple t in Q such that $\mathsf{vars}(t) \subseteq \mathsf{dom}(\mu)$ and $\mu(t) \in G_2 \setminus G_1$. A fragment is *e-witnessing* if so are all of its queries.

Informally, a query Q is e-witnessing if whenever an extension of a graph leads to a new answer, this answer is justified by a triple pattern in Q which maps to the extension. Unions of wwd-patterns can be shown e-witnessing. On the other hand, $\mathcal{U}$ is not e-witnessing, as can be seen on the pattern and graphs in Example 19. Hence, we obtain the following theorem.

▶ **Theorem 24.** *It holds that* $\mathcal{U}_{\text{wd}} < \mathcal{U}_{\text{wwd}} < \mathcal{U}$.

In contrast, queries over unions of wwd-patterns are as expressive as full SPARQL.

▶ **Theorem 25.** *It holds that* $\mathcal{S}_{\text{wwd}} \sim \mathcal{S}$.

As a consequence, every SPARQL query can be rewritten to a query over a union of "flat" patterns in depth-one normal form (Definition 10), albeit at the expense of a worst-case exponential blow-up in size.

7 Static Analysis of wwd-Patterns

In this section, we look at the general static analysis problems of query equivalence, containment, and subsumption. Formally, equivalence for a language $\mathcal{L}$ is defined as follows.

EQUIVALENCE($\mathcal{L}$)	**Input:**	Queries Q and Q' from $\mathcal{L}$
	Question:	Is $Q \equiv Q'$?

This problem is commonly generalised to CONTAINMENT($\mathcal{L}$), in which one checks whether Q is *contained* in Q', that is, whether $[\![Q]\!]_G \subseteq [\![Q']\!]_G$ holds for every graph G. We have $Q \equiv Q'$ if and only if Q and Q' contain each other. Furthermore, Letelier et al. [27] proposed the problem SUBSUMPTION($\mathcal{L}$), where one checks whether Q is *subsumed* by Q', that is, whether $[\![Q]\!]_G \sqsubseteq [\![Q']\!]_G$ holds for every G.

These problems have been studied for FILTER-free wd-patterns in [27, 33], establishing NP-completeness of equivalence and containment, and Π_2^p-completeness of subsumption. Moreover, all three problems are Π_2^p-complete for unions of FILTER-free wd-patterns, and undecidable for fragments with projection. Finally, from the results in [38] it follows that containment and subsumption are undecidable for $\mathcal{U}$. On the other hand, nothing seems to be known so far for well-designed patterns with FILTER.

We next show that equivalence, containment, and subsumption are all Π_2^p-complete for $\mathcal{P}_{\text{wwd}}$ and $\mathcal{U}_{\text{wwd}}$ (whereas $\mathcal{S}_{\text{wwd}}$ is undecidable by the results in [33]). The upper bound for containment follows from a small counterexample property: if $P \not\subseteq P'$ for some P and P' from $\mathcal{U}_{\text{wwd}}$, then there is a witnessing mapping of size $O(|P| + |P'|)$. Given this property, a

Table 1 Structure of query patterns in DBpedia logs.

	DBpedia 3.8			DBpedia 3.9		
	unique patterns	fraction of total	fraction of OPT	unique patterns	fraction of total	fraction of OPT
total	7 014 249	100%		27 854	100%	
patterns with OPT	742 002	10.58%	100%	1 639	5.83%	100%
unions of wd-patterns	238 995	3.41%	32.32%	972	3.49%	59.31%
unions of wwd-patterns	736 051	10.49%	99.19%	1 620	5.82%	98.84%

Π_2^p algorithm for containment is straightforward – we guess a mapping μ and a graph G of linear size, check that $\mu \notin [\![P']\!]_G$, and then call a coNP oracle for checking $\mu \in [\![P]\!]_G$. As a corollary, Equivalence($\mathcal{U}_{\text{wwd}}$) is also in Π_2^p. The argument for subsumption is analogous.

Hardness of subsumption and equivalence is established by a reduction from $\forall\exists$3SAT, while containment is Π_2^p-hard by the results in [33].

▶ **Theorem 26.** *Problems* Equivalence($\mathcal{L}$), Containment($\mathcal{L}$) *and* Subsumption($\mathcal{L}$) *are* Π_2^p*-complete for any* $\mathcal{L} \in \{\mathcal{P}_{\text{wwd}}, \mathcal{U}_{\text{wwd}}\}$.

Hence, for UNION- and FILTER-free patterns the step from well-designed to weakly well-designed OPT incurs a complexity jump for containment and equivalence. However, for the fragments with UNION or projection complexity remains the same in all three cases. As far as we are aware, these are the first decidability results on query equivalence and related problems for SPARQL fragments with OPT and FILTER.

8 Analysis of DBpedia Logs

In this section, we present a preliminary analysis of query logs over DBpedia, which suggests that the step from wd- to wwd-patterns makes a dramatic difference in real life: while only about half of the queries with OPT have well-designed patterns, almost all of these patterns fall into the weakly well-designed fragment.

DBpedia [26] is a project providing access to RDF data extracted from Wikipedia via a SPARQL endpoint. DBpedia query logs are well suited for analysing the structure of real-life SPARQL queries as they contain a large amount of general-purpose knowledge base queries, generated both manually and automatically. DBpedia query logs have been analysed by Picalausa and Vansummeren [32], who reported that, over a period in 2010, about 46.38% of a total of 1344K distinct DBpedia queries used OPT. However, only 47.80% of the queries with OPT had well-designed patterns. Another analysis of DBpedia logs from the USEWOD2011 data set performed by Arias Gallego et al. [9] concluded that 16.61% of about 5166K queries contain OPT; however, detailed structure of queries was not analysed.

We considered query logs over DBpedia 3.8 from USEWOD2013 [10] and DBpedia 3.9 logs from USEWOD2014 [11]. The DBpedia 3.8 set is a random selection of almost 12M queries from 2012 while the DBpedia 3.9 set contains only 253K queries, from 2013 and beginning of 2014. We removed syntactically incorrect queries as well as queries outside of $\mathcal{S}$ (in particular, queries using operators specific to SPARQL 1.1). Also, we rewrote the patterns of the remaining queries to unions of UNION-free patterns as proposed in [31] and eliminated duplicates, which left us with just over 7M queries over DBpedia 3.8 and 28K queries over DBpedia 3.9 (the decrease from 253K to 28K for DBpedia 3.9 is mostly due to duplicate elimination – with duplicates, we still have 197K queries). Finally, we isolated queries involving OPT and counted how many of their patterns were in $\mathcal{U}_{\text{wwd}}$ and in $\mathcal{U}_{\text{wd}}$.

The results are given in Table 1. They confirm that a non-negligible number of DBpedia queries use OPT; the exact fraction, however, varies considerably between the logs. In both cases, however, by far not all queries with OPT are well-designed (only 32% for DBpedia 3.8 and 59% for DBpedia 3.9), which is consistent with the results in [32]. On the other hand, almost all of the patterns with OPT (around 99% in both cases) are weakly well-designed, which we consider as the main practical justification for wwd-patterns.

9 Conclusion and Future Work

In this paper, we introduced a new fragment of SPARQL patterns called weakly well-designed patterns. This fragment extends the widely studied well-designed fragment by allowing variables from the optional side of an OPT-subpattern that are not "guarded" by the mandatory side to occur in certain positions outside of the subpattern. We showed that queries with wwd-patterns enjoy the same low complexity of evaluation as well-designed queries but cover almost all real-life queries. Moreover, our fragment is the maximal coNP fragment that does not impose structural restrictions on basic patterns and filter conditions. We studied the expressive power of the fragment and the complexity of its query optimisation problems.

For future work, we want to extend wwd-patterns to allow for non-top-level occurrences of UNION and projection. Also, we want to take into account features of SPARQL 1.1 [20] such as GRAPH, NOT EXISTS and property paths. Finally, we would like to implement our ideas in a prototype and compare its performance with existing SPARQL engines.

References

1 AllegroGraph. URL: `http://franz.com/agraph/allegrograph/`.
2 Apache Jena. URL: `http://jena.apache.org`.
3 RDF4J. URL: `http://rdf4j.org`.
4 Virtuoso Universal Server. URL: `http://virtuoso.openlinksw.com`.
5 Renzo Angles and Claudio Gutierrez. The expressive power of SPARQL. In *ISWC*, pages 114–129, 2008.
6 Marcelo Arenas, Sebastián Conca, and Jorge Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *WWW*, pages 629–638, 2012.
7 Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In *PODS*, pages 14–26, 2014.
8 Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *PODS*, pages 305–316, 2011.
9 Mario Arias Gallego, Javier D. Fernández, Miguel A. Martínez-Prieto, and Pablo de la Fuente. An empirical study of real-world SPARQL queries. In *USEWOD*, 2011. arXiv:1103.5043.
10 Bettina Berendt, Laura Hollink, Markus Luczak-Rösch, Knud Möller, and David Vallet. USEWOD2013: 3rd international workshop on usage analysis and the web of data. In *ESWC*, 2013.
11 Bettina Berendt, Laura Hollink, Markus Luczak-Rösch, Knud Möller, and David Vallet. USEWOD2014: 4th international workshop on usage analysis and the web of data. In *ESWC*, 2014.
12 Stefan Bischof, Markus Krötzsch, Axel Polleres, and Sebastian Rudolph. Schema-agnostic query rewriting in SPARQL 1.1. In *ISWC*, pages 584–600, 2014.

13 Carlos Buil-Aranda, Marcelo Arenas, and Oscar Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *ESWC*, pages 1–15. Springer, 2011.

14 Carlos Buil Aranda, Axel Polleres, and Jürgen Umbrich. Strategies for executing federated queries in SPARQL1.1. In *ISWC*, pages 390–405, 2014.

15 Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. SPARQL query containment under RDFS entailment regime. In *IJCAR*, pages 134–148, 2012.

16 Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. SPARQL query containment under SHI axioms. In *AAAI*, pages 10–16, 2012.

17 Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. URL: `http://www.w3.org/TR/rdf11-concepts/`.

18 Floris Geerts, Grigoris Karvounarakis, Vassilis Christophides, and Irini Fundulaki. Algebraic structures for capturing the provenance of SPARQL queries. In *ICDT*, pages 153–164, 2013.

19 Harry Halpin and James Cheney. Dynamic provenance for SPARQL updates. In *ISWC*, pages 425–440, 2014.

20 Steve Harris and Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, March 2013. URL: `http://www.w3.org/TR/sparql11-query/`.

21 Patrick J. Hayes and Peter F. Patel-Schneider. RDF 1.1 semantics. W3C recommendation, W3C, February 2014. URL: `http://www.w3.org/TR/rdf11-mt/`.

22 Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *ISWC*, pages 552–567, 2014.

23 Egor V. Kostylev and Bernardo Cuenca Grau. On the semantics of SPARQL queries with optional matching under entailment regimes. In *ISWC*, pages 374–389, 2014.

24 Egor V. Kostylev, Juan L. Reutter, Miguel Romero, and Domagoj Vrgoc. SPARQL with property paths. In *ICWC*, pages 3–18, 2015.

25 Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *ICDT*, pages 212–229, 2015.

26 Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

27 Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *ACM Transactions on Database Systems*, 38(4:25), 2013.

28 Katja Losemann and Wim Martens. The complexity of evaluating path expressions in SPARQL. In *PODS*, pages 101–112, 2012.

29 Frank Manola, Eric Miller, and Brian McBride. RDF 1.1 primer. W3C working group note, W3C, June 2014. URL: `http://www.w3.org/TR/rdf11-primer/`.

30 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. In *ISWC*, pages 30–43, 2006.

31 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3), 2009.

32 François Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *SWIM*, 2011.

33 Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed SPARQL. In *PODS*, pages 39–50, 2014.

34 Axel Polleres and Johannes Peter Wallner. On the relation between SPARQL1.1 and answer set programming. *Journal of Applied Non-Classical Logics*, 23(1-2):159–212, 2013.

35 Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C recommendation, W3C, January 2008. URL: `http://www.w3.org/TR/rdf-sparql-query/`.

36 Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In *ICDT*, pages 4–33, 2010.

37 Xiaowang Zhang and Jan Van den Bussche. On the primitivity of operators in SPARQL. *Information Processing Letters*, 114(9):480–485, 2014.

38 Xiaowang Zhang and Jan Van den Bussche. On the satisfiability problem for SPARQL patterns, 2014. arXiv:1406.1404.

39 Xiaowang Zhang and Jan Van den Bussche. On the power of SPARQL in expressing navigational queries. *The Computer Journal*, 58(11):2841–2851, 2015.

A Framework for Estimating Stream Expression Cardinalities

Anirban Dasgupta[1], Kevin J. Lang[2], Lee Rhodes[3], and Justin Thaler[4]

1 **IIT Gandhinagar, Gandhinagar, India**
anirban.dasgupta@gmail.com
2 **Yahoo Inc., 701 First Ave, Sunnyvale, CA, USA**
langk@yahoo-inc.com
3 **Yahoo Inc., 701 First Ave, Sunnyvale, CA, USA**
lrhodes@yahoo-inc.com
4 **Yahoo Inc., New York, NY, USA**
jthaler@fas.harvard.edu

Abstract

Given m distributed data streams $A_1, \ldots, A_m$, we consider the problem of estimating the number of unique identifiers in streams defined by set expressions over $A_1, \ldots, A_m$. We identify a broad class of algorithms for solving this problem, and show that the estimators output by any algorithm in this class are perfectly unbiased and satisfy strong variance bounds. Our analysis unifies and generalizes a variety of earlier results in the literature. To demonstrate its generality, we describe several novel sampling algorithms in our class, and show that they achieve a novel tradeoff between accuracy, space usage, update speed, and applicability.

1998 ACM Subject Classification G.3 [Probability and Statistics] Probabilistic algorithms, Stochastic processes, H.2.8 [Database Applications] Data mining

Keywords and phrases sketching, data stream algorithms, mergeability, distinct elements, set operations

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.6

1 Introduction

Consider an internet company that monitors the traffic flowing over its network by placing a sensor at each ingress and egress point. Because the volume of traffic is large, each sensor stores only a small *sample* of the observed traffic, using some simple sampling procedure. At some later point, the company decides that it wishes to estimate the number of unique users who satisfy a certain property P and have communicated over its network. We refer to this as the DistinctOnSubPopulation$_P$ problem, or Distinct$_P$ for short. How can the company combine the samples computed by each sensor, in order to accurately estimate the answer to this query?

In the case that P is the trivial property that is satisfied by all users, the answer to the query is simply the number of DistinctElements in the traffic stream, or Distinct for short. The problem of designing streaming algorithms and sampling procedures for estimating DistinctElements has been the subject of intense study. In general, however, P may be significantly more complicated than the trivial property, and may not be known until query time. For example, the company may want to estimate the number of (unique) men in a certain age range, from a specified country, who accessed a certain set of websites

during a designated time period, while excluding IP addresses belonging to a designated blacklist. This more general setting, where P is a nontrivial ad hoc property, has received somewhat less attention than the basic DISTINCT problem.

In this paper, our goal is to identify a simple method for combining the samples from each sensor, so that the following holds. As long as each sensor is using a sampling procedure that satisfies a certain mild technical condition, then for any property P, the combining procedure outputs an estimate for the DISTINCT$_P$ problem that is unbiased. Moreover, its variance should be bounded by that of the individual sensors' sampling procedures.[1]

For reasons that will become clear later, we refer to our proposed combining procedure as the *Theta-Sketch Framework*, and we refer to the mild technical condition that each sampling procedure must satisfy to guarantee unbiasedness as 1-*Goodness*. If the sampling procedures satisfy an additional property that we refer to as *monotonicity*, then the variance of the estimate output by the combining procedure is guaranteed to satisfy the desired variance bound. The Theta-Sketch Framework, and our analysis of it, unifies and generalizes a variety of results in the literature (see Section 2.5 for details).

The Importance of Generality. As we will see, there is a huge array of sampling procedures that the sensors could use. Each procedure comes with a unique tradeoff between accuracy, space requirements, update speed, and simplicity. Moreover, some of these procedures come with additional desirable properties, while others do not. We would like to support as many sampling procedures as possible, because the best one to use in any given given setting will depend on the relative importance of each resource in that setting.

Handling Set Expressions. The scenario described above can be modeled as follows. Each sensor observes a stream of identifiers A_j from a data universe of size n, and the goal is to estimate the number of distinct identifiers that satisfy property P in the combined stream $U = \cup_j A_j$. In full generality, we may wish to handle more complicated set expressions applied to the constituent streams, other than set-union. For example, we may have m streams of identifiers $A_1, \ldots, A_m$, and wish to estimate the number of distinct identifiers satisfying property P that appear in *all streams*. The Theta-Sketch Framework can be naturally extended to provide estimates for such queries. Our analysis applies to any sequence of set operations on the A_j's, but we restrict our attention to set-union and set-intersection throughout the paper for simplicity.

2 Preliminaries, Background, and Contributions

2.1 Notation and Assumptions

Streams and Set Operations. Throughout, A denotes a stream of identifiers from a data universe $[n] := \{1, \ldots, n\}$. We view any *property* P on identifiers as a subset of $[n]$, and let $n_{P,A} := \text{DISTINCT}_P(A)$ denote the number of distinct identifiers that appear in A and satisfy P. For brevity, we let n_A denote DISTINCT(A). When working in a multi-stream setting, $A_1, \ldots, A_m$ denote m streams of identifiers from $[n]$, $U := \cup_{j=1}^{m} A_j$ will denote the concatenation of the m input streams, while $I := \cap_{j=1}^{m} A_j$ denotes the set of identifiers that

[1] More precisely, we are interested in showing that the variance of the returned estimate is at most that of the (hypothetical) estimator obtained by running each individual sensor's sampling algorithm on the concatenated stream $A_1 \circ \cdots \circ A_m$. We refer to the latter estimator as "hypothetical" because it is typically infeasible to materialize the concatenated stream in distributed environments.

appear at least once in all m streams. Because we are interested only in *distinct* counts, it does not matter for definitional purposes whether we view U and I as sets, or as multisets. For any property $P\colon [n] \to \{0,1\}$, $n_{P,U} := \textsc{Distinct}_P(U)$ and $n_{P,I} := \textsc{Distinct}_P(I)$, while $n_U := \textsc{Distinct}(U)$ and $n_I := \textsc{Distinct}(I)$.

Hash Functions. For simplicity and clarity, and following prior work (e.g. [4, 5]), we assume throughout that the sketching and sampling algorithms make use of a perfectly random hash function h mapping the data universe $[n]$ to the open interval $(0,1)$. That is, for each $x \in [n]$, $h(x)$ is a uniform random number in $(0,1)$. Given a subset of hash values S computed from a stream A, and a property $P \subseteq [n]$, $P(S)$ denotes the subset of hash values in S whose corresponding identifiers in $[n]$ satisfy P. Finally, given a stream A, the notation X^{n_A} refers to the set of hash values obtained by mapping a hash function h over the n_A distinct identifiers in A.

2.2 Prior Art: Sketching Procedures for Distinct Queries

There is a sizeable literature on streaming algorithms for estimating the number of distinct elements in a single data stream. Some, but not all, of these algorithms can be modified to solve the $\textsc{Distinct}_P$ problem for general properties P. Depending on which functionality is required, systems based on HyperLogLog Sketches, K'th Minimum Value (KMV) Sketches, and Adaptive Sampling represent the state of the art for practical systems [11].[2] For clarity of exposition, and due to space constraints, we defer a more thorough overview of these algorithms to the full version of the paper [6]. Here, we briefly review the main concepts and relevant properties of each.

HLL: HyperLogLog Sketches. HLL is a sketching algorithm for the vanilla $\textsc{Distinct}$ problem. Its accuracy per bit is superior to the KMV and Adaptive Sampling algorithms described below. However, unlike KMV and Adaptive Sampling, it is not known how to extend the HLL sketch to estimate $n_{P,A}$ for general properties P (unless, of course, P is known prior to stream processing).

KMV: K'th Minimum Value Sketches. The KMV sketching procedure for estimating $\textsc{Distinct}(A)$ works as follows. While processing an input stream A, KMV keeps track of the set S of the k smallest unique hashed values of stream elements. The update time of a heap-based implementation of KMV is $O(\log k)$. The KMV estimator for $\textsc{Distinct}(A)$ is: $\text{KMV}_A = k/m_{k+1}$, where m_{k+1} denotes the $k{+}1^{\text{st}}$ smallest unique hash value.[3] It has been proved by [4], [10], and others, that $E(\text{KMV}_A) = n_A$, and $\sigma^2(\text{KMV}_A) = \frac{n_A^2 - k\, n_A}{k-1} < \frac{n_A^2}{k-1}$. Duffield et al. [7] proposed to change the heap-based implementation of the KMV sketching algorithm to an implementation based on quickselect [12]. This reduces the sketch update cost from $O(\log k)$ to amortized $O(1)$. However, this $O(1)$ hides a larger constant than competing methods. At the cost of storing the sampled identifiers, and not just their hash values, the KMV sketching procedure can be extended to estimate $n_{P,A}$ for any property $P \subseteq [n]$.

[2] Algorithms with better asymptotic bit-complexity are known [13], but they do not match the practical performance of the algorithms discussed here.

[3] Some works use the estimate k/m_k, *e.g.* [3]. We use k/m_{k+1} because it is unbiased, and for consistency with the work of Cohen and Kaplan [5] described below.

Adaptive Sampling. Adaptive Sampling maintains a sampling level $i \geq 0$, and the set S of all hash values less than 2^{-i}; whenever $|S|$ exceeds a pre-specified size limit, i is incremented and S is scanned discarding any hash value that is now too big. Because a simple scan is cheaper than running quickselect, an implementation of this scheme is typically faster than KMV. The estimator of n_A is $\text{Adapt}_A = |S|/2^{-i}$. It has been proved by [8] that this estimator is unbiased, and that $\sigma^2(\text{Adapt}_A) \approx 1.44(n_A^2/(k-1))$, where the approximation sign hides oscillations caused by the periodic culling of S. Like KMV, Adaptive Sampling can be extended to estimate $n_{P,A}$ for any property P. Although the stream processing speed of Adaptive Sampling is excellent, the fact that its accuracy oscillates as n_A increases is a shortcoming.

HLL for set operations on streams. HLL can be directly adapted to handle set-union. For set-intersection, the relevant adaptation uses the inclusion/exclusion principle. However, the variance of this estimate is approximately a factor of n_U/n_I worse than the variance achieved by the multiKMV algorithm described below. When $n_I \ll n_U$, this penalty factor overwhelms HLL's fundamentally good accuracy per bit.

KMV for set operations on streams. Given streams $A_1, \ldots, A_m$, let S_j denote the KMV sketch computed from stream A_j. A trivial way to use these sketches to estimate the number of distinct items n_U in the union stream U is to let M'_U denote the $(k+1)^{\text{st}}$ smallest value in the union of the sketches, and let $S'_U = \{x \in \cup_j S_j : x < M'_U\}$. Then S'_U is identical to the sketch that would have been obtained by running KMV directly on the concatenated stream $A_1 \circ \ldots, A_m$, and hence $\text{KMV}_{P,U} := k/M'_U$ is an unbiased estimator for n_U, by the same analysis as in the single-stream setting. We refer to this procedure as the "non-growing union rule."

Intuitively, the non-growing union rule does not use all of the information available to it. The sets S_j contain up to $k \cdot M$ distinct samples in total, but S'_U ignores all but the k smallest samples. With this in mind, Cohen and Kaplan [5] proposed the following adaptation of KMV to handle unions of multiple streams. We denote their algorithm by multiKMV, and also refer to it as the "growing union rule". Define $M_U = \min_{j=1}^m M_j$, and $S_U = \{x \in \cup_j S_j : x < M_U\}$. Then n_U is estimated by $\text{multiKMV}_U := |S_U|/M_U$, and $n_{P,U}$ is estimated by $\text{multiKMV}_{P,U} := |P(S_U)|/M_U$.

At first glance, it may seem obvious that the growing union rule yields an estimator that is "at least as good" as the non-growing union, since the growing union rule makes use of at least as many samples as the non-growing rule. However, it is by no means trivial to prove that $\text{multiKMV}_{P,U}$ is unbiased, nor that its variance is dominated by that of the non-growing union rule. Nonetheless, [5] managed to prove this: they showed that $\text{multiKMV}_{P,U}$ is unbiased and has variance that is dominated by the variance of $\text{KMV}_{P,U}$:

$$\sigma^2(\text{multiKMV}_{P,U}) \leq \sigma^2(\text{KMV}_{P,U}). \tag{1}$$

As observed in [5], multiKMV can be adapted in a similar manner to handle set-intersections (see Section 3.7 for details).

Adaptive Sampling for set operations on streams. Adaptive Sampling can handle set unions and intersections with a similar "growing union rule". Specifically, let $M_U := \min_{j=1}^m (2^{-i})_j$. Here, $(2^{-i})_j$ denotes the threshold for discarding hash values that was computed by the jth Adaptive Sampling sketch. We refer to this algorithm as multiAdapt. [9] proved epsilon-delta bounds on the error of $\text{multiAdapt}_{P,U}$, but did not derive expressions

for mean or variance. However, multiAdapt and multiKMV are both special cases of our Theta-Sketch Framework, and in Section 3 we will prove (apparently for the first time) that multiAdapt$_{P,U}$ is unbiased, and satisfies strong variance bounds. These results have the following two advantages over the epsilon-delta bounds of [9]. First, proving unbiasedness is crucial for obtaining estimators for distinct counts over subpopulations: these estimators are analyzed as a sum of a huge number of per-item estimates (see Theorem 11 for details), and biases add up. Second, variance bounds enable derivation of confidence intervals that an epsilon-delta guarantee cannot provide, unless the guarantee holds for many values of delta simultaneously.

2.3 Overview of the Theta-Sketch Framework

In this overview, we describe the Theta-Sketch Framework in the multi-stream setting where the goal is to output $n_{P,U}$, where $U = \cup_{j=1}^{m} A_j$ (we define the framework formally in Section 2.4). That is, the goal is to identify a very large class of sampling algorithms that can run on each constituent stream A_j, as well as a "universal" method for combining the samples from each A_j to obtain a good estimator for $n_{P,U}$. We clarify that the Theta-Sketch Framework, and our analysis of it, yields unbiased estimators that are interesting even in the single-stream case, where $m = 1$.

We begin by noting the striking similarities between the multiKMV and multiAdapt algorithms outlined in Section 2.2. In both cases, a sketch can be viewed as pair (θ, S) where θ is a certain threshold that depends on the stream, and S is a set of hash values which are all strictly less than θ. In this view, both schemes use the same estimator $|S|/\theta$, and also the same growing union rule for combining samples from multiple streams. The only difference lies in their respective rules for mapping streams to thresholds θ. The Theta-Sketch Framework formalizes this pattern of similarities and differences.

The assumed form of the single-stream sampling algorithms. The Theta-Sketch Framework demands that each constituent stream A_j be processed by a sampling algorithm samp$_j$ of the following form. While processing A_j, samp$_j$ evaluates a "threshold choosing function" (TCF) $T^{(j)}(A_j)$. The final state of samp$_j$ must be of the form $(\theta_j := T^{(j)}(A_j), S)$, where S is the set of all hash values strictly less than θ_j that were observed while processing A_j. If we want to estimate $n_{P,U}$ for non-trivial properties P, then samp$_j$ must also store the corresponding identifier that hashed to each value in S. Note that the framework itself does not specify the threshold-choosing functions $T^{(j)}$. Rather, any specification of the TCFs $T^{(j)}$ defines a particular instantiation of the framework.

▶ Remark. It might appear from Algorithm 1 that for any TCF $T^{(j)}$, the function samp$_j[T^{(j)}]$ makes two passes over the input stream: one to compute θ_j, and another to compute S_j. However, in all of the instantiations we consider, both operations can be performed in a single pass.

The universal combining rule. Given the states $(\theta_j := T^{(j)}(A_j), S_j)$ of each of the m sampling algorithms when run on the streams $A_1, \ldots, A_m$, define $\theta_U := \min_{j=1}^{m} \theta_j$, and $S_U := \{x \in \cup_j S_j : x < \theta_U\}$ (see the function ThetaUnion in Algorithm 1). Then n_U is estimated by $\hat{n}_U := |S_U|/\theta_U$, and $n_{P,U}$ as $\hat{n}_{P,U} := |P(S_U)|/\theta_U$ (see the function EstimateOnSubPopulation in Algorithm 1).

The analysis. Our analysis shows that, so long as each threshold-choosing function $T^{(j)}$ satisfies a mild technical condition that we call *1-Goodness*, then $\hat{n}_{P,U}$ is unbiased. We also

Algorithm 1 Theta Sketch Framework for estimating $n_{P,U}$. The framework is parameterized by choice of TCF's $T^{(j)}(k,A_j,h)$, one for each input stream.

1: **Definition:** Function $\text{samp}_j[T^{(j)}](k,\ A_j,\ h)$
  2: $\theta_j \leftarrow \mathrm{T}^{(j)}(k, A_j, h)$
  3: $S_j \leftarrow \{(x \in h(A_j)) < \theta_j\}$.
  4: **return** (θ_j, S_j).
5: **Definition:** Function ThetaUnion(Theta Sketches $\{(\theta_j, S_j)\}$)
  6: $\theta_U \leftarrow \min\{\theta_j\}$.
  7: $S_U \leftarrow \{(x \in (\cup S_j)) < \theta_U\}$.
  8: **return** (θ_U, S_U).
9: **Definition:** Function EstimateOnSubPopulation(Theta Sketch (θ, S) produced from stream A, Property P mapping identifiers to $\{0,1\}$)
  10: **return** $\hat{n}_{A,P} := \frac{|P(S)|}{\theta}$.

show that if each $T^{(j)}$ satisfies a certain additional condition that we call *monotonicity*, then $\hat{n}_{P,U}$ satisfies strong variance bounds (analogous to the bound of Equation (1) for KMV). Our analysis is arguably surprising, because 1-Goodness does not imply certain properties that have traditionally been considered important, such as permutation invariance, or S being a uniform random sample of the hashed unique items of the input stream.

Applicability. To demonstrate the generality of our analysis, we identify several valid instantiations of the Theta-Sketch Framework. First, we show that the TCF's used in KMV and Adaptive Sampling both satisfy 1-Goodness and monotonicity, implying that multiKMV and multiAdapt are both unbiased and satisfy the aforementioned variance bounds. For multiKMV, this is a reproof of Cohen and Kaplan's results [5], but for multiAdapt the results are new. Second, we identify a variant of KMV that we call pKMV, which is useful in multi-stream settings where the lengths of constituent streams are highly skewed. We show that pKMV satisfies both 1-Goodness and monotonicity. Third, we introduce a new sampling procedure that we call the *Alpha Algorithm*. Unlike earlier algorithms, the Alpha Algorithm's final state actually depends on the stream order, yet we show that it satisfies 1-Goodness, and hence is unbiased in both the single- and multi-stream settings. We also establish variance bounds on the Alpha Algorithm in the single-stream setting. We show experimentally that the Alpha Algorithm, in both the single- and multi-stream settings, achieves a novel tradeoff between accuracy, space usage, update speed, and applicability.

Unlike KMV and Adaptive Sampling, the Alpha Algorithm does not satisfy monotonicity in general. In fact, we have identified contrived examples in the multi-stream setting on which the aforementioned variance bounds are (weakly) violated. The Alpha Algorithm does, however, satisfy monotonicity under the promise that the $A_1, \ldots, A_m$ are pairwise disjoint, implying variance bounds in this case. Our experiments suggest that, in practice, the normalized variance in the multi-stream setting is not much larger than in the pairwise disjoint case.

Deployment of Algorithms. Within Yahoo, the pKMV and Alpha algorithms are used widely. In particular, stream cardinalities in Yahoo empirically satisfy a power law, with some very large streams and many short ones, and pKMV is an attractive option for such settings. We have released an optimized open-source implementation of our algorithms at `http://datasketches.github.io/`.

2.4 Formal Definition of Theta-Sketch Framework

The Theta-Sketch Framework is defined as follows. This definition is specific to the multi-stream setting where the goal is to output $n_{P,U}$, where $U = \cup_{j=1}^{m} A_j$ is the union of constituent streams $A_1, \ldots, A_m$.

▶ **Definition 1.** The Theta-Sketch Framework consists of the following components:

- The data type (θ, S), where $0 < \theta \leq 1$ is a threshold, and S is the set of all unique hashed stream items $0 \leq x < 1$ that are less than θ. We will generically use the term "theta-sketch" to refer to an instance of this data type.
- The universal "combining function" ThetaUnion(), defined in Algorithm 1, that takes as input a collection of theta-sketches (purportedly obtained by running samp[T]() on constituent streams $A_1, \ldots, A_m$), and returns a single theta-sketch (purportedly of the union stream $U = \cup_{i=1}^{m} A_i$).
- The function EstimateOnSubPopulation(), defined in Algorithm 1, that takes as input a theta-sketch (θ, S) (purportedly obtained from some stream A) and a property $P \subseteq [n]$ and returns an estimate of $\hat{n}_{P,A}$.

Any instantiation of the Theta-Sketch Framework must specify a "threshold choosing function" (TCF), denoted $T(k, A, h)$, that maps a target sketch size, a stream, and a hash function h to a threshold θ. Any TCF T implies a "base" sampling procedure samp[T]() that maps a target size, a stream A, and a hash function to a theta-sketch using the pseudocode shown in Algorithm 1. One can obtain an estimate $\hat{n}_{P,A}$ for $n_{P,A}$ by feeding the resulting theta-sketch into EstimateOnSubPopulation().

Given constituent streams $A_1, \ldots, A_m$, the instantiation obtains an estimate $\hat{n}_{P,U}$ of $n_{P,U}$ by running samp[T]() on each constituent stream A_j, feeding the resulting theta-sketches to ThetaUnion() to obtain a "combined" theta-sketch for $U = \cup_{i=1}^{m} A_i$, and then running EstimateOnSubPopulation() on this combined sketch.

▶ Remark. Definition 1 assumes for simplicity that the same TCF T is used in the base sampling algorithms run on each of the constituent streams. However, all of our results that depend only on 1-Goodness (*e.g.* unbiasedness of estimates and non-correlation of "per-item estimates") hold even if different 1-Good TCF's are used on each stream, and even if different values of k are employed.

2.5 Summary of Contributions

In summary, our contributions are: (1) Formulating the Theta-Sketch Framework. (2) Identifying a mild technical condition (1-Goodness) on TCF's ensuring that the framework's estimators are unbiased. (3) Identifying an additional mild technical condition (monotonicity) ensuring that the framework's estimators come with strong variance bounds analogous to Equation (1). (4) Introducing the pKMV Algorithm, a novel variant of multiKMV that can be useful in industrial big-data systems. (5) Proving that multiKMV, multiAdapt, and pKMV all satisfy 1-Goodness and monotonicity, implying unbiasedness and variance bounds for each. (6) Introducing the Alpha Algorithm, and proving that it satisfies 1-Goodness (thus implying unbiasedness), but not monotonicity. We also derive quantitative bounds on the Alpha Algorithm's variance in the single-stream setting, and present experimental evidence that it provides a novel tradeoff between accuracy, space usage, update speed, and applicability in both the single-stream and multi-stream settings.

3 Analysis of the Theta-Sketch Framework

Section Outline. Section 3.1 shows that KMV and Adaptive Sampling are both instantiations of the Theta-Sketch Framework. Section 3.2 defines 1-Goodness. Section 3.3 proves that the TCF's that instantiate behavior identical to KMV and Adapt both satisfy 1-Goodness. Section 3.4 proves that if a framework instantiation's TCF satisfies 1-Goodness, then so does the TCF that is implicitly applied to the union stream via the composition of the instantiation's base algorithm and the function ThetaUnion(). Section 3.5 proves that the estimator $\hat{n}_{P,A}$ for $n_{P,A}$ returned by EstimateOnSubPopulation() is unbiased when applied to any theta-sketch produced by a TCF satisfying 1-Goodness. Section 3.6 defines monotonicity and shows that 1-Goodness and monotonicity together imply variance bounds on $\hat{n}_{P,U}$. Section 3.7 explains how to tweak the Theta-Sketch Framework to handle set intersections and other set operations on streams.

3.1 Example Instantiations

Define m_{k+1} to be the $k{+}1^{\text{st}}$ smallest unique hash value in $h(A)$ (the hashed version of the input stream). The following is an easy observation.

▶ **Observation 2.** When the Theta-Sketch Framework is instantiated with the TCF $T(k, A, h) = m_{k+1}$, the resulting instantiation is equivalent to the multiKMV algorithm outlined in Section 2.2.

Let β be any real value in $(0, 1)$. For any z, define $\beta^{i(z)}$ to be the largest value of β^i (with i a non-negative integer) that is less than z.

▶ **Observation 3.** When the Theta-Sketch Framework is instantiated with the TCF $T(k, A, h) = \beta^{i(m_{k+1})}$ the resulting instantiation is equivalent to multiAdapt, which combines Adaptive Sampling with a growing union rule (cf. Section 2.2).[4]

3.2 Definition of 1-Goodness

The following circularity is a main source of technical difficulty in analyzing theta sketches: for any given identifier ℓ in a stream A, whether its hashed value $x_\ell = h(\ell)$ will end up in a sketch's sample set S depends on a comparison of x_ℓ versus a threshold $T(X^{n_A})$ that depends on x_ℓ itself. Adapting a technique from [5], we partially break this circularity by analyzing the following infinite family of projections of a given threshold choosing function $T(X^{n_A})$.

▶ **Definition 4** (Definition of Fix-All-But-One Projection)**.** Let T be a threshold choosing function. Let ℓ be one of the n_A unique identifiers in a stream A. Let $X^{n_A}_{-\ell}$ be a fixed assignment of hash values to all unique identifiers in A *except* for ℓ. Then the fix-all-but-one projection $T_\ell[X^{n_A}_{-\ell}](x_\ell) : (0, 1) \to (0, 1]$ of T is the function that maps values of x_ℓ to theta-sketch thresholds via the definition $T_\ell[X^{n_A}_{-\ell}](x_\ell) = T(X^{n_A})$, where X^{n_A} is the obvious combination of $X^{n_A}_{-\ell}$ and x_ℓ.

[5] analyzed similar projections under the assumption that the base algorithm is specifically (a weighted version of) KMV; we will instead impose the weaker condition that every fix-all-but-one projection satisfies 1-Goodness, defined below.[5]

---

[4] Section 2.2 assumed that the parameter β was set to the most common value: $1/2$.

[5] We chose the name 1-Goodness due to the reference to Fix-All-But-*One* Projections.

▶ **Definition 5** (Definition of 1-Goodness for Univariate Functions). A function $f(x) : (0,1) \to (0,1]$ satisfies 1-Goodness iff there exists a fixed threshold F such that:

$$\text{If } x < F, \text{ then } f(x) = F. \tag{2}$$

$$\text{If } x \geq F, \text{ then } f(x) \leq x. \tag{3}$$

▶ **Condition 6** (Definition of 1-Goodness for TCF's). A TCF $T(X^{n_A})$ satisfies 1-Goodness iff for every stream A containing n_A unique identifiers, every label $\ell \in A$, and every fixed assignment $X^{n_A}_{-\ell}$ of hash values to the identifiers in $A \setminus \ell$, the fix-all-but-one projection $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell)$ satisfies Definition 5.

3.3 TCF's of multiKMV and multiAdapt Both Satisfy 1-Goodness

The following two easy theorems show that the Threshold Choosing Functions used respectively in KMV and in Adaptive Sampling both satisfy the 1-Goodness condition.

▶ **Theorem 7.** *If $T(X^{n_A}) = m_{k+1}$, then every fix-all-but-one projection $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell)$ of T satisfies 1-Goodness.*

Proof. Let $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell)$ be any specific fix-all-but-one-projection of $T(X^{n_A}) = m_{k+1}$. We will exhibit the fixed value $F_\ell[X^{n_A}_{-\ell}]$ that causes (2) and (3) to be true for this projection. Let a and b respectively be the k'th and $(k{+}1)^{\text{st}}$ smallest hash values in $X^{n_A}_{-\ell}$. Then Subconditions (2) and (3) hold for $F_\ell[X^{n_A}_{-\ell}] = a$. There are three cases:

Case $(x_\ell < a < b)$: In this case, $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = T(X^{n_A}) = m_{k+1} = a$. Since $x_\ell < (F_\ell[X^{n_A}_{-\ell}] = a)$, (2) holds because $(\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = a) = F_\ell[X^{n_A}_{-\ell}]$, and (3) holds vacuously.

Case $(a < x_\ell < b)$: In this case, $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = T(X^{n_A}) = m_{k+1} = x_\ell$. Since $x_\ell \geq (F_\ell[X^{n_A}_{-\ell}] = a)$, (3) holds because $(\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = x_\ell) \leq x_\ell$, and (2) holds vacuously.

Case $(a < b < x_\ell)$: In this case, $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = T(X^{n_A}) = m_{k+1} = b$. Since $x_\ell \geq (F_\ell[X^{n_A}_{-\ell}] = a)$, (3) holds because $(\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = b) < x_\ell$, and (2) holds vacuously.

◀

▶ **Theorem 8.** *If $T(X^{n_A}) = \beta^{i(m_{k+1})}$, then every fix-all-but-one projection $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell)$ of T satisfies 1-Goodness.*

Proof. Let $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell)$ be any specific fix-all-but-one-projection of $T(X^{n_A}) = \beta^{i(m_{k+1})}$. We will exhibit the fixed value $F_\ell[X^{n_A}_{-\ell}]$ that causes (2) and (3) to be true for this projection. Let a and b respectively be the k'th and $(k{+}1)^{\text{st}}$ smallest hash values in $X^{n_A}_{-\ell}$. Then Subconditions (2) and (3) hold for $F_\ell[X^{n_A}_{-\ell}] = \beta^{i(a)}$. There are four cases:

Case $(x_\ell < \beta^{i(a)} < a < b)$: $m_{k+1} = a$, so $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = \beta^{i(a)}$. Since $x_\ell < F_\ell[X^{n_A}_{-\ell}] = \beta^{i(a)}$, (2) holds because $(\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = \beta^{i(a)}) = F_\ell[X^{n_A}_{-\ell}]$, and (3) holds vacuously.

Case $(\beta^{i(a)} < x_\ell < a < b)$: $m_{k+1} = a$, so $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = \beta^{i(a)}$. Since $x_\ell \geq F_\ell[X^{n_A}_{-\ell}] = \beta^{i(a)}$, (3) holds because $(\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = \beta^{i(a)}) < x_\ell$, and (2) holds vacuously.

Case $(\beta^{i(a)} < a < x_\ell < b)$: $m_{k+1} = x_\ell$, so $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = \beta^{i(x_\ell)}$. Since $x_\ell \geq F_\ell[X^{n_A}_{-\ell}] = \beta^{i(a)}$, (3) holds because $(\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = \beta^{i(x_\ell)}) < x_\ell$, and (2) holds vacuously.

Case $\beta^{i(a)} < a < b < x_\ell)$: $m_{k+1} = b$, so $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = \beta^{i(b)}$. Since $x_\ell \geq F_\ell[X^{n_A}_{-\ell}] = \beta^{i(a)}$, (3) holds because $(\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell) = \beta^{i(b)}) < b < x_\ell$, and (2) holds vacuously.

◀

3.4 1-Goodness Is Preserved by the Function ThetaUnion()

Next, we show that if a framework instantiation's TCF T satisfies 1-Goodness, then so does the TCF T^U that is implicitly being used by the theta-sketch construction algorithm defined by the composition of the instantiation's base sampling algorithms and the function ThetaUnion(). We begin by formally extending the definition of a fix-all-but-one projection to cover the degenerate case where the label ℓ isn't actually a member of the given stream A.

▶ **Definition 9.** Let A be a stream containing n_A identifiers. Let ℓ be a label that is *not* a member of A. Let the notation $X^{n_A}_{-\ell}$ refer to an assignment of hash value to *all* identifiers in A. For any hash value x_ℓ of the non-member label ℓ, define the value of the "fix-all-but-one" projection $T_\ell[X^{n_A}_{-\ell}](x_\ell)$ to be the constant $T(X^{n_A}_{-\ell})$.

▶ **Theorem 10.** *If the threshold choosing functions $T^{(j)}(X^{n_{A_j}})$ of the base algorithms used to create sketches of m streams A_j all satisfy Condition 6, then so does the TCF:*

$$T^U(X^{n_U}) = \min_j\{T^{(j)}(X^{n_{A_j}})\} \tag{4}$$

that is implicitly applied to the union stream via the composition of those base algorithms and the procedure ThetaUnion*().*

Proof. Let $\mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$ be any specific fix-all-but-one projection of the threshold choosing function $T^U(X^{n_U})$ defined by Equation (4). We will exhibit the fixed value $F^U[X^{n_U}_{-\ell}]$ that causes (2) and (3) to be true for $\mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$.

The projection $\mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$ is specified by a label $\ell \in (A_U = \cup_j A_j)$, and a set $X^{n_U}_{-\ell}$ of fixed hash values for the identifiers in $A_U \backslash \ell$. For each j, those fixed hash values $X^{n_U}_{-\ell}$ induce a set $X^{n_{A_j}}_{-\ell}$ of fixed hash values for the identifiers in $A_j \backslash \ell$. The combination of ℓ and $X^{n_{A_j}}_{-\ell}$ then specifies a projection $\mathrm{T}^{(\mathrm{j})}_\ell[X^{n_{A_j}}_{-\ell}](x_\ell)$ of $T^{(j)}(X^j)$. Now, if $\ell \in A_j$, this is a fix-all-but-one projection according to the original Definition 4, and according to the current theorem's pre-condition, this projection must satisfy 1-Goodness for univariate functions. On the other hand, if $\ell \notin A_j$, this is a fix-all-but-one projection according to the extended Definition 9, and is therefore a constant function, and therefore satisfies 1-Goodness. Because the projection $\mathrm{T}^{(\mathrm{j})}_\ell[X^{n_{A_j}}_{-\ell}](x_\ell)$ satisfies 1-Goodness either way, there must exist a fixed value $F^j[X^{n_{A_j}}_{-\ell}]$ such that Subconditions (2) and (3) are true for $\mathrm{T}^{(\mathrm{j})}_\ell[X^{n_{A_j}}_{-\ell}](x_\ell)$.
We now show that the value $\mathrm{F}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}] := \min_j(\mathrm{F}^{\mathrm{j}}_\ell[X^{n_{A_j}}_{-\ell}])$ causes Subconditions (2) and (3) to be true for the projection $\mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$, thus proving that this projection satisfies 1-Goodness.

To show: $x_\ell < \mathrm{F}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}]$ implies $\mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell) = \mathrm{F}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}]$. The condition $x_\ell < \mathrm{F}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}]$ implies that for all j, $x_\ell < \mathrm{F}^{\mathrm{j}}_\ell[X^{n_{A_j}}_{-\ell}]$. Then, for all j, $\mathrm{T}^{(\mathrm{j})}_\ell[X^{n_{A_j}}_{-\ell}](x_\ell) = \mathrm{F}^{\mathrm{j}}_\ell[X^{n_{A_j}}_{-\ell}]$ by Subcondition (2) for the various $\mathrm{T}^{(\mathrm{j})}_\ell[X^{n_{A_j}}_{-\ell}](x_\ell)$. Therefore, $\mathrm{F}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}] = \min_j(\mathrm{F}^{\mathrm{j}}_\ell[X^{n_{A_j}}_{-\ell}]) = \min_j(\mathrm{T}^{(\mathrm{j})}_\ell[X^{n_{A_j}}_{-\ell}](x_\ell)) = \mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$, where the last step is by Eqn (4). This establishes Subcondition (2) for the projection $\mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$.

To show: $x_\ell \geq \mathrm{F}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}]$ implies $x_\ell \geq \mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$. Because x_ℓ is greater than or equal to $\mathrm{F}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}] = \min_j(\mathrm{F}^{\mathrm{j}}_\ell[X^{n_{A_j}}_{-\ell}])$, there exists a j such that $x_\ell \geq \mathrm{F}^{\mathrm{j}}_\ell[X^{n_{A_j}}_{-\ell}]$. By Subcondition (3) for this $\mathrm{T}^{(\mathrm{j})}_\ell[X^{n_{A_j}}_{-\ell}](x_\ell)$, we have $x_\ell \geq \mathrm{T}^{(\mathrm{j})}_\ell[X^{n_{A_j}}_{-\ell}](x_\ell)$. By Eqn (4), we then have $x_\ell \geq \mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$, thus establishing Subcondition (3) for $\mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$.

Finally, because the above argument applies to every projection $\mathrm{T}^{\mathrm{U}}_\ell[X^{n_U}_{-\ell}](x_\ell)$ of $T^U(X^{n_U})$, we have proved the desired result that $T^U(X^{n_U})$ satisfies Condition 6. ◀

3.5 Unbiasedness of EstimateOnSubPopulation()

We now show that 1-Goodness of a TCF implies that the corresponding instantiation of the Theta-Sketch Framework provides unbiased estimates of the number of unique identifiers on a stream or on the union of multiple streams.

▶ **Theorem 11.** *Let A be a stream containing n_A unique identifiers, and let P be a property evaluating to 1 on an arbitrary subset of the identifiers. Let h denote a random hash function. Let T be a threshold choosing function that satisfies Condition 6. Let (θ, S_A) denote a sketch of A created by $\text{samp}[T](k, A, h)$, and as usual let $P(S_A)$ denote the subset of hash values in S_A whose corresponding identifiers satisfy P. Then $\mathrm{E}_h(\hat{n}_{P,A}) := \mathrm{E}_h\left(\frac{|P(S_A)|}{\theta}\right) = n_{P,A}$.*

Theorems 10 and 11 together imply that, in the multi-stream setting, the estimate $\hat{n}_{P,U}$ for $n_{P,U}$ output by the Theta-Sketch Framework is unbiased, assuming the base sampling schemes $\text{samp}_j()$ each use a TCF $T^{(j)}$ satisfying 1-Goodness.

Proof. Let A be a stream, and let T be a Threshold Choosing Function that satisfies 1-Goodness. Fix any $\ell \in A$. For any assignment X^{n_A} of hash values to identifiers in A, define the "per-identifier estimate" V_ℓ as follows:

$$V_\ell(X^{n_A}) = \frac{S_\ell(X^{n_A})}{T(X^{n_A})} \quad \text{where} \quad S_\ell(X^{n_A}) = \begin{cases} 1 \text{ if } x_\ell < T(X^{n_A}) \\ 0 \text{ otherwise.} \end{cases} \tag{5}$$

Because T satisfies 1-Goodness, there exists a fixed threshold $F(X^{n_A}_{-\ell})$ for which it is a straightforward exercise to verify that:

$$V_\ell(X^{n_A}) = \begin{cases} 1/F(X^{n_A}_{-\ell}) \text{ if } x_\ell < F(X^{n_A}_{-\ell}) \\ 0 \text{ otherwise.} \end{cases} \tag{6}$$

Now, conditioning on $X^{n_A}_{-\ell}$ and taking the expectation with respect to x_ℓ:

$$E(V_\ell | X^{n_A}_{-\ell}) = \int_0^1 V_\ell[X^{n_A}](x_\ell)dx_\ell = F(X^{n_A}_{-\ell}) \cdot \frac{1}{F(X^{n_A}_{-\ell})} = 1. \tag{7}$$

Since Equation (7) establishes that $E(V_\ell) = 1$ when conditioned on each $X^{n_A}_{-\ell}$, we also have $E(V_\ell) = 1$ when the expectation is taken over all X^{n_A}. By linearity of expectation, we conclude that $E(\hat{n}_{P,A}) = \sum_{\ell \in A : P(\ell)=1} E(V_\ell) = n_{P,A}$. ◀

3.6 1-Goodness and Monotonicity Imply Variance Bound

As usual, let $U = \cup_{i=1}^m A_i$ be the union of m data streams. Our goal in this section is to identify conditions on a threshold choosing function which guarantee the following: whenever the Theta-Sketch Framework is instantiated with a TCF T satisfying the conditions, then for any property $P \subseteq [n]$, the variance $\sigma^2(\hat{n}_{P,U})$ of the estimator obtained from the Theta-Sketch Framework is bounded above by the variance of the estimator obtained by running $\text{samp}[T]()$ on the stream $A^* := A_1 \circ A_2 \circ \cdots \circ A_m$ obtained by concatenating $A_1, \ldots, A_m$.

It is easy to see that 1-Goodness alone is not sufficient to ensure such a variance bound. Consider, for example, a TCF T that runs KMV on a stream A unless it determines that $n_A \geq C$, for some fixed value C, at which point it sets θ to 1 (thereby causing $\text{samp}[T]()$ to sample all elements from A). Note that such a base sampling algorithm is not implementable by a sublinear space streaming algorithm, but T nonetheless satisfies 1-Goodness. It is easy to see that such a base sampling algorithm will fail to satisfy our desired comparative

variance result when run on constituent streams $A_1, \ldots, A_m$ satisfying $n_{A_i} < C$ for all i, and $n_U > C$. In this case, the variance of $\hat{n}_U$ will be positive, while the variance of the estimator obtained by running samp$[T]$ directly on A^* will be 0.

Thus, for our comparative variance result to hold, we assume that T satisfies both 1-Goodness and the following additional monotonicity condition.

▶ **Condition 12** (Monotonicity Condition)**.** Let A_0, A_1, A_2 be any three streams, and let $A^* := A_0 \circ A_1 \circ A_2$ denote their concatenation. Fix any hash function h and parameter k. Let $\theta = T(k, A_1, h)$, and $\theta' = T(k, A^*, h)$. Then $\theta' \leq \theta$.

▶ **Theorem 13.** *Suppose that the Theta-Sketch Framework is instantiated with a TCF T that satisfies Condition 6 (1-Goodness), as well as Condition 12 (monotonicity). Fix a property P, and let $A_1, \ldots A_m$, be m input streams. Let $U = \cup A_j$ denote the union of the distinct labels in the input streams. Let $A^* = A_1 \circ A_2 \circ \ldots \circ A_m$ denote the concatenation of the input streams. Let $(\theta^*, S^*) =$ samp$[T](k, A^*, h)$, and let $\hat{n}^{A^*}_{P,A^*}$ denote the estimate of $n_{P,A^*} = n_{P,U}$ obtained by evaluating* EstimateOnSubPopulation$((\theta^*, S^*), P)$*. Let $(\theta^U, S^U) =$ ThetaUnion$(\{(\theta_j, S_j)\})$, and let $\hat{n}^U_{P,U}$ denote the estimate of $n_{P,U} = n_{P,A^*}$ obtained by evaluating* EstimateOnSubPopulation$((\theta^U, S^U), P)$*. Then, with the randomness being over the choice of hash function h, $\sigma^2(\hat{n}^U_{P,U}) \leq \sigma^2(\hat{n}^{A^*}_{P,A^*})$.*

The proof of Theorem 13 is rather involved, and is deferred to the full version of the paper.

On the applicability of Theorem 13. It is easy to see that Condition 12 holds for any TCF that is (1) order-insensitive and (2) has the property that adding another distinct item to the stream cannot increase the resulting threshold θ. The TCF T used in multiKMV (namely, $T(k, A, h) = m_{k+1}$), satisfies these properties, as does the TCF used in Adaptive Sampling. Since we already showed that both of these TCF's satisfy 1-Goodness, Theorem 13 applies to multiKMV and multiAdapt. In Section 4, we introduce the pKMV algorithm, which is useful in multi-stream settings where the distribution of stream lengths is highly skewed, and we show that Theorem 13 applies to this algorithm as well.

In Section 5, we introduce the Alpha Algorithm and show that it satisfies 1-Goodness. While the Alpha Algorithm does not satisfy monotonicity in general, it *does* under the promise that $A_1, \ldots, A_m$ are pairwise disjoint; Theorem 13 applies in this case. Our experiments (deferred to the full version of the paper) suggest that, in practice, the normalized variance in the multi-stream setting is not much larger than in the pairwise disjoint case.

3.7 Handling Set Intersections

The Theta-Sketch Framework can be tweaked in a natural way to handle set intersection and other set operations, just as was the case for multiKMV. Specifically, define $\theta_U = \min_{j=1}^m \theta_j$, and $S_I = \{(x \in \cap_j S_j) < \theta_U\}$. The estimator for $n_{P,I}$ is $\hat{n}_{P,I} := |P(S_I)|/\theta_U$.

It is not difficult to see that $\hat{n}_{P,I}$ is exactly equal to $\hat{n}_{P',U}$, where P' is the property that evaluates to 1 on an identifier if and only if the identifier satisfies P and is also in I. Since the latter estimator was already shown to be unbiased with variance bounded as per Theorem 13, $\hat{n}_{P,I}$ satisfies the same properties.

4 The pKMV Variant of KMV

Motivation. An internet company involved in online advertising typically faces some version of the following problem: there is a huge stream of events representing visits of users to

web pages, and a huge number of relevant "profiles", each defined by the combination of a predicate on users and a predicate on web pages. On behalf of advertisers, the internet company must keep track of the count of distinct users who generate events that match each profile. The distribution (over profiles) of these counts typically is highly skewed and covers a huge dynamic range, from hundreds of millions down to just a few.

Because the summed cardinalities of all profiles is huge, the brute force technique (of maintaining, for each profile, a hash table of distinct user ids) would use an impractical amount of space. A more sophisticated approach would be to run multiKMV, treating each profile as separate stream A_i. This effectively replaces each hash table in the brute force approach with a KMV sketch. The problem with multiKMV in this setting is that, while KMV does avoid storing the entire data stream for streams containing more than k distinct identifiers, KMV produces no space savings for streams shorter than k. Because the vast majority of profiles contain only a few users, replacing the hash tables in the brute force approach by KMV sketches might still use an impractical amount of space.

On the other hand, fixed-threshold sampling with $\theta = p$ for a suitable sampling rate p, would always result in an expected factor $1/p$ saving in space, relative to storing the entire input stream. However, this method may result in too large a sample rate for long streams (i.e., for profiles satisfied by many users), also resulting in an impractical amount of space.

The pKMV algorithm. In this scenario, the hybrid Threshold Choosing Function $T(k, A, h) = \min(m_{k+1}, p)$ can be a useful compromise, as it ensures that even short streams get downsampled by a factor of p, while long streams produce at most k samples. While it is possible to prove that this TCF satisfies 1-Goodness via a direct case analysis, the property can also be established by an easier argument: Consider a hypothetical computation in which the ThetaUnion procedure is used to combine two sketches of the same input stream: one constructed by KMV with parameter k, and one constructed by fixed-threshold sampling with parameter p. Clearly, this computation outputs $\theta = \min(m_{k+1}, p)$. Also, since KMV and fixed-threshold sampling both satisfy 1-Goodness, and ThetaUnion preserves 1-Goodness (cf. Theorem 11), T also satisfies 1-Goodness.

It is easy to see that Condition 12 applies to $T(k, A, h) = \min(m_{k+1}, p)$ as well. Indeed, T is clearly order-insensitive, so it suffices to show that adding an additional identifier to the stream cannot increase the resulting threshold. Since p never changes, the only way that adding another distinct item to the stream could increase the threshold would be by increasing m_{k+1}. However, that cannot happen.

5 Alpha Algorithm

5.1 Motivation and Comparison to Prior Art

Section 3's theoretical results are strong because they cover such a wide class of base sampling algorithms. In fact, 1-Goodness even covers base algorithms that lack certain traditional properties such as invariance to permutations of the input, and uniform random sampling of the input. We are now going to take advantage of these strong theoretical results for the Theta Sketch Framework by devising a novel base sampling algorithm that lacks those traditional properties, but still satisfies 1-Goodness. Our main purpose for describing our Alpha Algorithm in detail is to exhibit the generality of the Theta-Sketch Framework. Nonetheless the Alpha Algorithm does have the following advantages relative to HLL, KMV, and Adaptive Sampling.

Advantages over HLL. Unlike HLL, the Alpha Algorithm provides unbiased estimates for $\textsc{Distinct}_P$ queries for non-trivial predicates P. Also, when instantiating the Theta-Sketch Framework via the Alpha Algorithm in the multi-stream setting, the error behavior scales better than HLL for general set operations (cf. Section 2.2). Finally, because the Alpha Algorithm computes a sample, its output is human-interpretable and amenable to post-processing.

Advantages over KMV. Implementations of KMV must either use a heap data structure or quickselect [12] to give quick access to the $k+1^{\text{st}}$ smallest unique hash value seen so far. The heap-based implementation yields $O(\log k)$ update time, and quickselect, while achieving $O(1)$ update time, hides a large constant factor in the Big-Oh notation (cf. Section 2.2). The Alpha Algorithm avoids the need for a heap or quickselect, yielding superior practical performance.

Advantages over Adaptive Sampling. The accuracy of Adaptive Sampling oscillates as n_A increases. The Alpha Algorithm avoids this behavior.

The remainder of this section provides a detailed analysis of the Alpha Algorithm. In particular, we show that it satisfies 1-Goodness, and we give quantitative bounds on its variance in the single-stream setting. The full version of the paper describes experiments showing that, in both the single- and multi-stream settings, the Alpha Algorithm achieves a novel tradeoff between accuracy, space usage, update speed, and applicability.

5.2 AlphaTCF

Algorithm 2 describes the threshold choosing function AlphaTCF. AlphaTCF can be viewed as a tightly interleaved combination of two different processes. One process uses the set D to remove duplicate items from the raw input stream; the other process uses a technique similar to Approximate Counting [14] to estimate the number of items in the de-duped stream created by the first process. In addition, the second process maintains and frequently reduces a threshold $\theta = \alpha^i$ that is used by the first process to identify hash values that *cannot* be members of S, and therefore don't need to be placed in the de-duping set D. If the set D is implemented using a standard dynamically-resized hash table, then well-known results imply that the amortized cost[6] of processing each stream element is $O(1)$, and the space occupied by the hash table is $O(|D|)$. However, there is a simple optimized implementation of the Alpha Algorithm, based on Cuckoo Hashing, that implicitly, and at zero cost, deletes all members of D that are not less than θ, and therefore are not members of S (see the full version of the paper for details). This does not affect correctness, because those deleted members will not be needed for future de-duping tests of hash values that will all be less than θ. Furthermore, in Theorem 15 below, it is proved that $|S|$ is tightly concentrated around k. Hence, the space usage of this optimized implementation is $O(k)$ with probability $1 - o(1)$.

5.3 AlphaTCF Satisfies 1-Goodness

We will now prove that AlphaTCF satisfies 1-Goodness, thus implying unbiasedness.

▶ **Theorem 14.** *If $T(X^{n_A}) = AlphaTCF$, then every fix-all-but-one projection $\mathrm{T}_\ell[X^{n_A}_{-\ell}](x_\ell)$ of $T(X^{n_A})$ satisfies 1-Goodness.*

[6] Recent theoretical results imply that the update time can be made worst-case $O(1)$ [1, 2].

Algorithm 2 The Alpha Algorithm's Threshold Choosing Function

```
1: Function AlphaTCF (target size k, stream A, hash function h)
2: α ← k/(k + 1).
3: prefix(h(A)) ← shortest prefix of h(A) containing exactly k unique hash values.
4: suffix(h(A)) ← the corresponding suffix.
5: D ← the set of unique hash values in prefix(h(A)).
6: i ← 0.
7: for all x ∈ suffix(h(A)) do
8:    if x < α^i then
9:       if x ∉ D then
10:         i ← i + 1.
11:         D ← D ∪ {x}.
12:      end if
13:   end if
14: end for
15: return θ ← α^i.
```

Proof. Fix the number of distinct identifiers n_A in A. Consider any identifier ℓ appearing in the stream, and let $x = h(\ell)$ be its hash value. Fix the hash values of all other elements of the sequence of values $X_{-\ell}^{n_A}$. We need to exhibit a threshold F such that $x < F$ implies $T_\ell[X_{-\ell}^{n_A}](x_\ell)(x) = F$ and $x \geq F$ implies $T_\ell[X_{-\ell}^{n_A}](x) \leq x$.

First, if x lies in one of the first $k + 1$ positions in the stream, then $T_\ell[X_{-\ell}^{n_A}](x)$ is a constant independent of x; in this case, F can be set to that constant.

Now for the main case, suppose that ℓ does not lie in one of the first $k + 1$ positions of the stream. Consider a subdivision of the hashed stream into the initial segment preceding $x = h(\ell)$, then x itself, then the final segment that follows x. Because all hash values besides x are fixed in $X_{-\ell}^{n_A}$, during the initial segment, there is a specific number a of times that θ is decreased. When x is processed, θ is decreased either zero or one times, depending on whether $x < \alpha^a$. Then, during the final segment, θ will be decreased a certain number of additional times, where this number depends on whether $x < \alpha^a$. Let b denote the number of additional times θ is decreased if $x < \alpha^a$, and c the number of additional times θ is decreased otherwise. This analysis is summarized in the following table:

Rule	Condition on x	Final value of θ
L	$x < \alpha^a$	α^{a+b+1}
G	$x \geq \alpha^a$	α^{a+c+0}

We prove the theorem using the threshold $F = \alpha^{a+b+1}$. We note that $F = \alpha^{a+b+1} < \alpha^a$, so F and α^a divide the range of x into three disjoint intervals, creating three cases that need to be considered.

Case 1: $x < F < \alpha^a$. In this case, because $x < F$, we need to show that $T_\ell[X_{-\ell}^{n_A}](x) = F$. By Rule L, $T_\ell[X_{-\ell}^{n_A}](x) = \alpha^{a+b+1} = F$.

Case 2: $F \leq x < \alpha^a$. Because $x \geq F$, we need to show that $T_\ell[X_{-\ell}^{n_A}](x) \leq x$. By Rule L, $T_\ell[X_{-\ell}^{n_A}](x) = \alpha^{a+b+1} = F \leq x$.

Case 3: $F < \alpha^a \leq x$. Because $x \geq F$, we need to show that $T_\ell[X_{-\ell}^{n_A}](x) \leq x$. By Rule G, $T_\ell[X_{-\ell}^{n_A}](x) = \alpha^{a+c+0} \leq \alpha^a \leq x$. ◀

5.4 Analysis of Alpha Algorithm on Single Streams

The following two theorems show that the Alpha Algorithm's space usage and single-stream estimation accuracy are quite similar to those of KMV. That means that it is safe to use the Alpha Algorithm as a drop-in replacement for KMV in a sketching-based big-data system,

which then allows the system to benefit from the Alpha Algorithm's low update cost. See the experiments in the full version of the paper for an empirical comparison of these costs.

Random Variables. When Line 15 of Algorithm 2 is reached after processing a randomly hashed stream, the program variable i is governed by a random variable $\mathcal{I}$. Similarly, when Line 3 of Algorithm 1 is subsequently reached, the cardinality of the set S is governed by a random variable $\mathcal{S}$. The following two theorems characterize the distributions of $\mathcal{S}$ and of the Theta Sketch Framework's estimator $\mathcal{S}/(\alpha^{\mathcal{I}})$. Specifically, Theorem 15 shows that the number of elements sampled by the Alpha Algorithm is tightly concentrated around k, and hence its space usage is concentrated around that of KMV. Theorem 16 shows that the variance of the estimate returned by the Alpha Algorithm is very close to that of KMV. Their proofs are deferred to the full version of the paper.

▶ **Theorem 15.** *Let $\mathcal{S}$ denote the cardinality of the set S computed by the Alpha Algorithm's Threshold Choosing Function (Algorithm 2). Then:*

$$\mathrm{E}(\mathcal{S}) = k. \tag{8}$$

$$\sigma^2(\mathcal{S}) < \frac{k}{2} + \frac{1}{4}. \tag{9}$$

▶ **Theorem 16.** *Let $\mathcal{S}$ denote the cardinality of the set S computed by the Alpha Algorithm's Threshold Choosing Function (Algorithm 2). Then:*

$$\sigma^2(\mathcal{S}/(\alpha^{\mathcal{I}})) = \frac{(2k+1)n_A^2 - (k^2+k)(2n_A - 1) - n_A}{2k^2} \tag{10}$$

$$< \frac{n_A^2}{k - \frac{1}{2}}. \tag{11}$$

5.5 Variance of the Alpha Algorithm in the Multi-Stream Setting

The Alpha Algorithm does not satisfy monotonicity (Condition 12) in general, so Theorem 13 does not immediately imply variance bounds in the multi-stream setting. In fact, we have identified contrived examples in the multi-stream setting on which the variance of the Theta-Sketch Framework when instantiated with the TCF of the Alpha Algorithm is slightly larger than the hypothetical estimator obtained by running the Alpha Algorithm on the concatenated stream $A_1 \circ \ldots A_m$ (the worst-case setting appears to be when $A_1 \ldots A_m$ are all permutations of each other).

However, we show in this section that the Alpha Algorithm does satisfy monotonicity under the promise that all constituent streams are pairwise disjoint. This implies the variance guarantees of Theorem 13 do apply to the Alpha Algorithm under the promise that $A_1, \ldots, A_m$ are pairwise disjoint. Our experiments suggest that, in practice, the normalized variance of the Alpha Algorithm in the multi-stream setting is not much larger than in the pairwise disjoint case.

▶ **Theorem 17.** *The TCF computed by the Alpha Algorithm satisfies Condition 12 under the promise that the streams A_1, A_2, A_3 appearing in Condition 12 are pairwise disjoint.*

Proof. Due to space constraints, the proof is deferred to the full version of the paper. ◀

References

1. Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 107–118, 2009. doi:10.1007/978-3-642-02927-1_11.
2. Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 787–796, 2010. doi:10.1109/FOCS.2010.80.
3. Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Cambridge, MA, USA, September 13-15, 2002, Proceedings*, pages 1–10, 2002. doi:10.1007/3-540-45726-7_1.
4. Kevin S. Beyer, Rainer Gemulla, Peter J. Haas, Berthold Reinwald, and Yannis Sismanis. Distinct-value synopses for multiset operations. *Commun. ACM*, 52(10):87–95, 2009. doi:10.1145/1562764.1562787.
5. Edith Cohen and Haim Kaplan. Leveraging discarded samples for tighter estimation of multiple-set aggregates. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems, SIGMETRICS/Performance 2009, Seattle, WA, USA, June 15-19, 2009*, pages 251–262, 2009. doi:10.1145/1555349.1555379.
6. Anirban Dasgupta, Kevin Lang, Lee Rhodes, and Justin Thaler. A framework for estimating stream expression cardinalities. *CoRR*, abs/1510.01455, 2015. URL: http://arxiv.org/abs/1510.01455.
7. Nick G. Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 54(6), 2007. doi:10.1145/1314690.1314696.
8. Philippe Flajolet. On adaptive sampling. *Computing*, 43(4):391–400, 1990. doi:10.1007/BF02241657.
9. Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *SPAA*, pages 281–291, 2001. doi:10.1145/378580.378687.
10. Frédéric Giroire. Order statistics and estimating cardinalities of massive data sets. *Discrete Applied Mathematics*, 157(2):406–427, 2009. doi:10.1016/j.dam.2008.06.020.
11. Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In *Joint 2013 EDBT/ICDT Conferences, EDBT'13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 683–692, 2013. doi:10.1145/2452376.2452456.
12. C. A. R. Hoare. Algorithm 65: Find. *Commun. ACM*, 4(7):321–322, July 1961. doi:10.1145/366622.366647.
13. Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *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 41–52, 2010. doi:10.1145/1807085.1807094.
14. Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, October 1978. doi:10.1145/359619.359627.

Declarative Probabilistic Programming with Datalog

Vince Barany[1], Balder ten Cate[2], Benny Kimelfeld[3], Dan Olteanu[4], and Zografoula Vagena[5]

1 **LogicBlox, Atlanta, GA, USA***
2 **LogicBlox, Atlanta, GA, USA†**
3 **Technion, Haifa, Israel; and LogicBlox, Atlanta, GA, USA**
4 **University of Oxford, Oxford, UK; and LogicBlox, Atlanta, GA, USA**
5 **LogicBlox, Atlanta, GA, USA**

Abstract

Probabilistic programming languages are used for developing statistical models, and they typically consist of two components: a specification of a stochastic process (the prior), and a specification of observations that restrict the probability space to a conditional subspace (the posterior). Use cases of such formalisms include the development of algorithms in machine learning and artificial intelligence. We propose and investigate an extension of Datalog for specifying statistical models, and establish a declarative probabilistic-programming paradigm over databases. Our proposed extension provides convenient mechanisms to include common numerical probability functions; in particular, conclusions of rules may contain values drawn from such functions. The semantics of a program is a probability distribution over the possible outcomes of the input database with respect to the program. Observations are naturally incorporated by means of integrity constraints over the extensional and intensional relations. The resulting semantics is robust under different chases and invariant to rewritings that preserve logical equivalence.

1998 ACM Subject Classification D.1.6 Logic Programming, G.3 Probability and Statistics, H.2.3 Languages

Keywords and phrases Chase, Datalog, probability measure space, probabilistic programming

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.7

1 Introduction

Languages for specifying general statistical models are commonly used in the development of machine learning and artificial intelligence algorithms for tasks that involve inference under uncertainty. A substantial effort has been made on developing such formalisms and corresponding system implementations. An actively studied concept in that area is that of *Probabilistic Programming* (PP) [20], where the idea is that the programming language allows for specifying general random procedures, while the system *executes* the program not in the standard programming sense, but rather by means of *inference*. Hence, a PP system is built around a language and an (approximate) inference engine, which typically makes use of Markov Chain Monte Carlo methods (e.g., the Metropolis-Hastings algorithm). The

* Now at Google, Inc.
† Now at Google, Inc.

relevant inference tasks can be viewed as probability-aware aggregate operations over all possible worlds, that is, possible outcomes of the program. Examples of such tasks include finding the most likely possible world, or estimating the probability of an event. Recently, DARPA initiated the project *Probabilistic Programming for Advancing Machine Learning (PPAML)*, aimed at advancing PP systems (with a focus on a specific collection of systems, e.g., [40, 30, 32]) towards facilitating the development of algorithms and software that are based on machine learning.

In probabilistic programming, a statistical model is typically phrased by means of two components. The first component is a *generative process* that produces a random possible world by straightforwardly following instructions with randomness, and in particular, sampling from common numerical probability functions; this gives the *prior distribution*. The second component allows to phrase constraints that the relevant possible worlds should satisfy, and, semantically, transforms the prior distribution into the *posterior distribution* – the subspace obtained by conditioning on the constraints.

As an example, in *supervised text classification* (e.g., spam detection) the goal is to classify a text document into one of several known classes (e.g., spam/non-spam). Training data consists of a collection of documents labeled with classes, and the goal of learning is to build a model for predicting the classes of unseen documents. One common approach to this task assumes a generative process that produces random *parameters* for every class, and then uses these parameters to define a generator of random words in documents of the corresponding class [33, 31]. The prior distribution thus generates parameters and documents for each class, and the posterior is defined by the actual documents of the training data. In *unsupervised* text classification the goal is to cluster a given set of documents, so that different clusters correspond to different topics (not known in advance). Latent Dirichlet Allocation [10] approaches this problem in a similar generative way as the above, with the addition that each document is associated with a distribution over topics.

A Datalog program is a set of logical rules, interpreted in the context of a relational database (where database relations are also called the *extensional relations*), that are used to define additional relations (known as the *intensional relations*). Datalog has traditionally been used as a database query language. In recent years, however, it has found new applications in data integration, information extraction, networking, program analysis, security, cloud computing, and enterprise software development [23]. In each of these applications, being declarative, Datalog makes specifications easier to write (sometimes with orders-of-magnitude fewer lines of code than imperative code, e.g., [28]), and to comprehend and maintain.

In this work, we extend Datalog with the ability to program statistical models. In par with existing languages for PP, our proposed extension consists of two parts: a *generative Datalog program* that specifies a prior probability space over (finite or infinite) sets of facts that we call *possible outcomes*, and a definition of the posterior probability by means of *observations*, which come in the form of ordinary logical constraints over the extensional and intensional relations. We subscribe to the premise of the PP community (and PPAML in particular) that this paradigm has the potential of substantially facilitating the development of applications that involve machine learning for inferring missing or uncertain information. Indeed, probabilistic variants are explored for the major programming languages, such as C [37], Java [27], Scala [40], Scheme [30] and Python [38] (we discuss the relationship of this work to related literature in Section 6). At LogicBlox, we are interested in extending our Datalog-based LogiQL [22] with PP to enable and facilitate the development of predictive analysis [6]. We believe that, once the semantics becomes clear, Datalog can offer a natural and appealing basis for PP, since it has an inherent (and well studied) separation between given data (EDB), generated data (IDB), and conditioning (constraints).

The main challenge, when attempting to extend Datalog with probabilistic programming constructs, is to retain the inherent features of Datalog. Specifically, the semantics of Datalog does not depend on the order by which the rules are resolved (chased). Hence, it is safe to provide a Datalog engine with the ability to decide on the chasing order that is estimated to be more efficient. Another feature is invariance under logical equivalence: two Datalog programs have the same semantics whenever their rules are equivalent when viewed as theories in first-order logic. Hence, it is safe for a Datalog engine to rewrite a program, as long as logical equivalence is preserved.

For example, consider an application where we want to predict the number of visits of clients to some local service (e.g., a doctor's office). For simplicity, suppose that we have a schema with the following relations: LivesIn(person, city), WorksIn(person, employer), LocatedIn(company, city), and AvgVisits(city, avg). The following rule provides an appealing way to model the generation of a random number of visits for a person.

$$\text{Visits}(p, \mathsf{Poisson}[\lambda]) \leftarrow \text{LivesIn}(p, c), \text{AvgVisits}(c, \lambda) \tag{1}$$

The conclusion of this rule involves sampling values from a parameterized probability distribution. Next, suppose that we do not have all the addresses of persons, and we wish to expand the simulation with employer cities. Then we might use the following additional rule.

$$\text{Visits}(p, \mathsf{Poisson}[\lambda]) \leftarrow \text{WorksIn}(p, e), \text{LocatedIn}(e, c), \text{AvgVisits}(c, \lambda) \tag{2}$$

Now, it is not clear how to interpret the semantics of Rules (1) and (2) in a manner that retains the declarative nature of Datalog. If, for a person p, the right sides of both rules are true, should both rules "fire" (i.e., should we sample the Poisson distribution twice)? And if p works in more than one company, should we have one sample per company? And if p lives in one city but works in another, which rule should fire? If only one rule fires, then the semantics becomes dependent on the chase order. To answer these questions, we need to properly define what it means for the head of a rule to be *satisfied* when it involves randomness such as $\mathsf{Poisson}[\lambda]$.

Furthermore, consider the following (standard) rewriting of the above program.

$$\begin{aligned}
\text{PersonCity}(p, c) &\leftarrow \text{LivesIn}(p, c)\\
\text{PersonCity}(p, c) &\leftarrow \text{WorksIn}(p, e), \text{LocatedIn}(e, c)\\
\text{Visits}(p, \mathsf{Poisson}[\lambda]) &\leftarrow \text{PersonCity}(p, c), \text{AvgVisits}(c, \lambda)
\end{aligned}$$

As a conjunction of first-order sentences, the rewritten program is equivalent to the previous one; we would therefore like the two programs to have the same semantics. In rule-based languages with a factor-based semantics, such as *Markov Logic Networks* [15] or *Probabilistic Soft Logic* [11], the above rewriting may change the semantics dramatically.

We introduce *PPDL*, a purely declarative probabilistic programming language based on Datalog. The generative component of a PPDL program consists of rules extended with constructs to refer to conventional parameterized numerical probability functions (e.g., Poisson, geometrical, etc.). Specifically, these mechanisms allow sampling values from the given parameterized distributions in the conclusion of a rule (and if desired, use these values as parameters of other distributions). In this paper, our focus is on discrete numerical distributions (the framework we introduce admits a natural generalization to continuous distributions, such as Gaussian or Pareto, but we defer the details of this to future work). Semantically, a PPDL program associates to each input instance I a probability distribution over *possible outcomes*. In the case where all the possible outcomes are finite, we get a discrete probability distribution, and the probability of a possible outcome can be defined immediately

from its content. But in general, a possible outcome can be infinite, and moreover, the set of all possible outcomes can be uncountable. Hence, in the general case we obtain a probability measure space. We define a natural notion of a *probabilistic chase* where existential variables are produced by invoking the corresponding numerical distributions. We define a measure space based on a chase, and prove that this definition is robust, in the sense that the same probability measure is obtained no matter which chase order is used.

A short version of this paper has appeared in the 2015 Alberto Mendelzon International Workshop [47].

2 Preliminaries

In this section we give basic notation and definitions that we use throughout the paper.

Schemas and instances. A (*relational*) *schema* is a collection $\mathcal{S}$ of *relation symbols*, where each relation symbol R is associated with an *arity*, denoted $\mathsf{arity}(R)$, which is a natural number. An *attribute* of a relation symbol R is any number in $\{1, \ldots, \mathsf{arity}(R)\}$. For simplicity, we consider here only databases over real numbers; our examples may involve strings, which we assume are translatable into real numbers. A *fact* over a schema $\mathcal{S}$ is an expression of the form $R(c_1, \ldots, c_n)$ where R is an n-ary relation in $\mathcal{S}$ and $c_1, \ldots, c_n \in \mathbb{R}$. An *instance* I over $\mathcal{S}$ is a finite set of facts over $\mathcal{S}$. We denote by R^I the set of all tuples $(c_1, \ldots, c_n)$ such that $R(c_1, \ldots, c_n) \in I$.

Datalog programs. PPDL extends Datalog without the use of existential quantifiers. However, we will make use of existential rules indirectly in the definition of the semantics. For this reason, we review here Datalog as well as existential Datalog. Formally, an *existential Datalog program*, or Datalog$^\exists$ *program*, is a triple $\mathcal{D} = (\mathcal{E}, \mathcal{I}, \Theta)$ where: (1) $\mathcal{E}$ is a schema, called the *extensional database* (EDB) schema, (2) $\mathcal{I}$ is a schema, called the *intensional database* (IDB) schema, disjoint from $\mathcal{E}$, and (3) Θ is a finite set of Datalog$^\exists$ *rules*, that is, first-order formulas of the form $\forall \mathbf{x}\big[\ (\exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})) \leftarrow \varphi(\mathbf{x})\ \big]$ where $\varphi(\mathbf{x})$ is a conjunction of atomic formulas over $\mathcal{E} \cup \mathcal{I}$ and $\psi(\mathbf{x}, \mathbf{y})$ is an atomic formula over $\mathcal{I}$, such that each variable in $\mathbf{x}$ occurs in φ. Here, by an *atomic formula* (or, *atom*) we mean an expression of the form $R(t_1, \ldots, t_n)$ where R is an n-ary relation and $t_1, \ldots, t_n$ are either constants (i.e., numbers) or variables. For readability's sake, we omit the universal quantifier and the parentheses around the conclusion (left-hand side), and write simply $\exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}) \leftarrow \varphi(\mathbf{x})$. *Datalog* is the fragment of Datalog$^\exists$ where the conclusion of each rule is an atomic formula without existential quantifiers.

Let $\mathcal{D} = (\mathcal{E}, \mathcal{I}, \Theta)$ be a Datalog$^\exists$ program. An *input instance* for $\mathcal{D}$ is an instance I over $\mathcal{E}$. A *solution* of I w.r.t. $\mathcal{D}$ is a possibly infinite set F of facts over $\mathcal{E} \cup \mathcal{I}$, such that $I \subseteq F$ and F satisfies all rules in Θ (viewed as first-order sentences). A *minimal solution* of I (w.r.t. $\mathcal{D}$) is a solution F of I such that no proper subset of F is a solution of I. The set of all, finite and infinite, minimal solutions of I w.r.t. $\mathcal{D}$ is denoted by $\mathsf{min\text{-}sol}_\mathcal{D}(I)$, and the set of all *finite* minimal solutions is denoted by $\mathsf{min\text{-}sol}^{\mathsf{fin}}_\mathcal{D}(I)$. It is a well known fact that, if $\mathcal{D}$ is a *Datalog program* (that is, without existential quantifiers), then every input instance I has a unique minimal solution, which is finite, and therefore $\mathsf{min\text{-}sol}^{\mathsf{fin}}_\mathcal{D}(I) = \mathsf{min\text{-}sol}_\mathcal{D}(I)$.

Probability spaces. We separately consider *discrete* and *continuous* probability spaces. We initially focus on the discrete case; there, a *probability space* is a pair (Ω, π), where Ω is a finite or countably infinite set, called the *sample space*, and $\pi : \Omega \to [0, 1]$ is such that

$\sum_{o\in\Omega}\pi(o)=1$. If (Ω,π) is a probability space, then π is a *probability distribution* over Ω. We say that π is a *numerical* probability distribution if $\Omega\subseteq\mathbb{R}$. In this work we focus on discrete numerical distributions.

A *parameterized* probability distribution is a function $\delta:\Omega\times\mathbb{R}^k\to[0,1]$, such that $\delta(\cdot,\mathbf{p}):\Omega\to[0,1]$ is a probability distribution for all $\mathbf{p}\in\mathbb{R}^k$. We use $\mathsf{pardim}(\delta)$ to denote the parameter dimension k. For presentation's sake, we may write $\delta(o|\mathbf{p})$ instead of $\delta(o,\mathbf{p})$. Moreover, we denote the (non-parameterized) distribution $\delta(\cdot|\mathbf{p})$ by $\delta[\mathbf{p}]$. An example of a parameterized distribution is $\mathsf{Flip}(\cdot|p)$, where Ω is $\{0,1\}$, and for a parameter $p\in[0,1]$ we have $\mathsf{Flip}(1|p)=p$ and $\mathsf{Flip}(0|p)=1-p$. Another example is $\mathsf{Poisson}(\cdot|\lambda)$, where $\Omega=\mathbb{N}$, and for a parameter $\lambda\in(0,\infty)$ we have $\mathsf{Poisson}(x|\lambda)=\lambda^x e^{-\lambda}/x!$. In Section 7 we discuss the extension of our framework to models that have a variable number of parameters, and to continuous distributions.

Let Ω be a set. A *σ-algebra* over Ω is a collection $\mathcal{F}$ of subsets of Ω, such that $\mathcal{F}$ contains Ω and is closed under complement and countable unions. (Implied properties include that $\mathcal{F}$ contains the empty set, and that $\mathcal{F}$ is closed under countable intersections.) If $\mathcal{F}'$ is a nonempty collection of subsets of Ω, then the closure of $\mathcal{F}'$ under complement and countable unions is a σ-algebra, and it is said to be *generated* by $\mathcal{F}'$. A *probability measure space* is a triple $(\Omega,\mathcal{F},\pi)$, where: *(1)* Ω is a set, called the *sample space*, *(2)* $\mathcal{F}$ is a σ-algebra over Ω, *and (3)* $\pi:\mathcal{F}\to[0,1]$, called a *probability measure*, is such that $\pi(\Omega)=1$, and $\pi(\cup\mathcal{E})=\sum_{e\in\mathcal{E}}\pi(e)$ for every countable set $\mathcal{E}$ of pairwise-disjoint elements of $\mathcal{F}$.

3 Generative Datalog

A Datalog program without existential quantifiers specifies how to obtain a minimal solution from an input instance by producing the set of inferred IDB facts. In this section we present *generative Datalog programs*, which specify how to infer a *distribution over possible outcomes* given an input instance. In Section 5 we will complement generative programs with *constraints* to establish the PPDL framework.

3.1 Syntax

The syntax of a generative Datalog program is defined as follows.

▶ **Definition 1** (GDatalog[Δ]). Let Δ be a finite set of parameterized numerical distributions.

1. A Δ-*term* is a term of the form $\delta[\![p_1,\ldots,p_k]\!]$ where $\delta\in\Delta$ is a parameterized distribution with $\mathsf{pardim}(\delta)=\ell\leq k$, and each p_i $(i=1,\ldots,k)$ is a variable or a constant. To improve readability, we will use a semicolon to separate the first ℓ arguments (corresponding to the distribution parameters) from the optional other arguments (which we will call the *event signature*), as in $\delta[\![\mathbf{p};\mathbf{q}]\!]$. When the event signature is empty (i.e., when $k=\ell$), we write $\delta[\![\mathbf{p};]\!]$.[1]
2. A Δ-*atom* in a schema $\mathcal{S}$ is an atomic formula $R(t_1,\ldots,t_n)$ with $R\in\mathcal{S}$ an n-ary relation, such that exactly one term t_i $(i=1,\ldots,n)$ is a Δ-term and the other terms t_j are variables and/or constants.[2]

[1] Intuitively, $\delta[\![\mathbf{p};\mathbf{q}]\!]$ denotes a sample from the distribution $\delta(\cdot|\mathbf{p})$ where different samples are drawn for different values of the event signature $\mathbf{q}$ (cf. Example 2).

[2] The restriction to at most one Δ-term per atom is only for presentational purposes, cf Section 3.5.

House	
id	*city*
NP1	Napa
NP2	Napa
YC1	Yucaipa

Business	
id	*city*
NP3	Napa
YC1	Yucaipa

City	
name	*burglaryrate*
Napa	0.03
Yucaipa	0.01

AlarmOn
unit
NP1
YC1
YC2

Figure 1 Input instance I of the burglar example.

1. $\mathrm{Earthquake}(c, \mathsf{Flip}[\![0.01; \mathsf{Earthquake}, c]\!]) \leftarrow \mathrm{City}(c, r)$
2. $\mathrm{Unit}(h, c) \leftarrow \mathrm{House}(h, c)$
3. $\mathrm{Unit}(b, c) \leftarrow \mathrm{Business}(b, c)$
4. $\mathrm{Burglary}(x, c, \mathsf{Flip}[\![r; \mathsf{Burglary}, x, c]\!]) \leftarrow \mathrm{Unit}(x, c)\,,\ \mathrm{City}(c, r)$
5. $\mathrm{Trig}(x, \mathsf{Flip}[\![0.6; \mathsf{Trig}, x]\!]) \leftarrow \mathrm{Unit}(x, c)\,,\ \mathrm{Earthquake}(c, 1)$
6. $\mathrm{Trig}(x, \mathsf{Flip}[\![0.9; \mathsf{Trig}, x]\!]) \leftarrow \mathrm{Burglary}(x, c, 1)$
7. $\mathrm{Alarm}(x) \leftarrow \mathrm{Trig}(x, 1)$

Figure 2 GDatalog[Δ] program $\mathcal{G}$ for the burglar example.

3. A *GDatalog*[Δ] *rule* over a pair of disjoint schemas $\mathcal{E}$ and $\mathcal{I}$ is a first-order sentence of the form $\forall \mathbf{x}(\psi(\mathbf{x}) \leftarrow \phi(\mathbf{x}))$ where $\phi(\mathbf{x})$ is a conjunction of atoms in $\mathcal{E} \cup \mathcal{I}$ and $\psi(\mathbf{x})$ is either an atom in $\mathcal{I}$ or a Δ-atom in $\mathcal{I}$.
4. A *GDatalog*[Δ] *program* is a triple $\mathcal{G} = (\mathcal{E}, \mathcal{I}, \Theta)$, where $\mathcal{E}$ and $\mathcal{I}$ are disjoint schemas and Θ is a finite set of GDatalog[Δ] rules over $\mathcal{E}$ and $\mathcal{I}$.

▶ **Example 2.** Our example is based on the burglar example of Pearl [39] that has been frequently used to illustrate probabilistic programming (e.g., [36]). Consider the EDB schema $\mathcal{E}$ consisting of the following relations: $\mathrm{House}(h, c)$ represents houses h and their location cities c, $\mathrm{Business}(b, c)$ represents businesses b and their location cities c, $\mathrm{City}(c, r)$ represents cities c and their associated burglary rates r, and $\mathrm{AlarmOn}(x)$ represents units (houses or businesses) x where the alarm is on. Figure 1 shows an instance I over this schema. Now consider the GDatalog[Δ] program $\mathcal{G} = (\mathcal{E}, \mathcal{I}, \Theta)$ of Figure 2.

Here, Δ consists of only one distribution, namely Flip. The first rule in Figure 2, intuitively, states that, for every fact of the form $\mathrm{City}(c, r)$, there must be a fact $\mathrm{Earthquake}(c, y)$ where y is drawn from the Flip (Bernoulli) distribution with the parameter 0.01. Moreover, the additional arguments Earthquake and c given after the semicolon (where Earthquake is a constant) enforce that different samples are drawn from the distribution for different cities (even if they have the same burglary rate), and that we never use the same sample as in Rules 5 and 6. Similarly, the presence of the additional argument x in Rule 4 enforces that a different sample is drawn for a different unit, instead of sampling only once per city.

▶ **Example 3.** The program of Figure 3 models virus dissemination among computers of email users. For simplicity, we identify each user with a distinct computer. Every message has a probability of passing a virus, if the virus is active on the source. If a message passes the virus, then the recipient has the virus (but it is not necessarily active, e.g., since the computer has the proper defence). And every user has a probability of having the virus active on her computer, in case she has the virus. Our program has the following EDBs:

- $\mathrm{Message}(m, s, t)$ contains message identifiers m sent from the user s to the user t.
- $\mathrm{VirusSource}(x)$ contains the users who are known to be virus sources.

1. $\text{PassVirus}(m, \mathsf{Flip}[\![0.1; m]\!]) \leftarrow \text{Message}(m, s, t), \text{ActiveVirus}(s, 1)$
2. $\text{HasVirus}(t) \leftarrow \text{PassVirus}(m, 1), \text{Message}(m, s, t)$
3. $\text{ActiveVirus}(x, \mathsf{Flip}[\![0.5; x]\!]) \leftarrow \text{HasVirus}(x)$
4. $\text{ActiveVirus}(x, 1) \leftarrow \text{VirusSource}(x)$

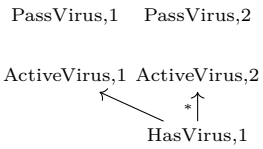


Figure 3 Program and dependency graph for the virus-dissemination example.

1. $\text{Earthquake}(c, \mathsf{Flip}[0.01]) \leftarrow \text{City}(c, r)$
2. $\text{Unit}(h, c) \leftarrow \text{House}(h, c)$
3. $\text{Unit}(b, c) \leftarrow \text{Business}(b, c)$
4. $\text{Burglary}(x, c, \mathsf{Flip}[r]) \leftarrow \text{Unit}(x, c), \text{City}(c, r)$
5. $\text{Trig}(x, \mathsf{Flip}[0.6]) \leftarrow \text{Unit}(x, c), \text{Earthquake}(c, 1)$
6. $\text{Trig}(x, \mathsf{Flip}[0.9]) \leftarrow \text{Burglary}(x, c, 1)$
7. $\text{Alarm}(x) \leftarrow \text{Trig}(x, 1)$

Figure 4 Burglar program from Figure 2 modified to use syntactic sugar.

In addition, the following IDBs are used.

- $\text{PassVirus}(m, b)$ determines whether a message m passes a virus ($b = 1$) or not ($b = 0$).
- $\text{HasVirus}(x, b)$ determines whether user x has the virus ($b = 1$) or not ($b = 0$).
- $\text{ActiveVirus}(x, b)$ determines whether user x has the virus active ($b = 1$) or not ($b = 0$).

The dependency graph depicted in Figure 3 will be used later on, in Section 3.4, when we further analyse this program.

Syntactic sugar. The syntax of GDatalog[Δ], as defined above, requires us to always make explicit the arguments that determine when different samples are taken from a distribution (cf. the argument c after the semicolon in Rule 1 of Figure 2, and the arguments x, c after the semicolon in Rule 4 of the same program). To enable a more succinct notation, we use the following convention: consider a Δ-atom $R(t_1, \ldots, t_n)$ in which the i-th argument, t_i, is a Δ-term. Then t_i may be written using the simpler notation $\delta[\mathbf{p}]$, in which case it is understood to be a shorthand for $\delta[\![\mathbf{p}; \mathbf{q}]\!]$ where $\mathbf{q}$ is the sequence of terms $\mathsf{r}, i, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$. Here, r is a constant uniquely associated to the relation R. Thus, for example, $\text{Earthquake}(c, \mathsf{Flip}[0.01]) \leftarrow \text{City}(c, r)$ is taken to be a shorthand for $\text{Earthquake}(c, \mathsf{Flip}[\![0.01; \mathsf{Earthquake}, 2, c]\!]) \leftarrow \text{City}(c, r)$. Using this syntactic sugar, the program in Figure 2 can be rewritten in a notationally less verbose way, cf. Figure 4. Note, however, that the shorthand notation is less explicit as to describing when two rules involve the same sample vs. different samples from the same probability distribution.

3.2 Possible Outcomes

A GDatalog[Δ] program $\mathcal{G} = (\mathcal{E}, \mathcal{I}, \Theta)$ is associated with a corresponding Datalog$^{\exists}$ program $\widehat{\mathcal{G}} = (\mathcal{E}, \mathcal{I}^{\Delta}, \Theta^{\Delta})$. The *possible outcomes* of an input instance I w.r.t. $\mathcal{G}$ will then be minimal solutions of I w.r.t. $\widehat{\mathcal{G}}$. Next, we describe $\mathcal{I}^{\Delta}$ and Θ^{Δ}.

The schema $\mathcal{I}^{\Delta}$ extends $\mathcal{I}$ with the following additional relation symbols: for each $\delta \in \Delta$ with $\mathsf{pardim}(\delta) = k$ and for each $n \geq 0$, we have a $(k + n + 1)$-ary relation symbol $\text{Result}^{\delta}_{n}$. These relation symbols $\text{Result}^{\delta}_{n}$ are called the *distributional* relation symbols of $\mathcal{I}^{\Delta}$, and the

1a. $\exists y\ \text{Result}_2^{\mathsf{Flip}}(0.01, \mathsf{Earthquake}, c, y) \leftarrow \text{City}(c, r)$
1b. $\text{Earthquake}(c, y) \leftarrow \text{City}(c, r), \text{Result}_2^{\mathsf{Flip}}(0.01, \mathsf{Earthquake}, c, y)$
2. $\text{Unit}(h, c) \leftarrow \text{House}(h, c)$
3. $\text{Unit}(b, c) \leftarrow \text{Business}(b, c)$
4a. $\exists y\ \text{Result}_3^{\mathsf{Flip}}(r, \mathsf{Burglary}, x, c, y) \leftarrow \text{Unit}(x, c)\,,\ \text{City}(c, r)$
4b. $\text{Burglary}(x, c, y) \leftarrow \text{Unit}(x, c)\,,\ \text{City}(c, r), \text{Result}_3^{\mathsf{Flip}}(r, \mathsf{Burglary}, x, c, y)$
5a. $\exists y \text{Result}_2^{\mathsf{Flip}}(0.6, \mathsf{Trig}, x, y) \leftarrow \text{Unit}(x, c)\,,\ \text{Earthquake}(c, 1)$
5b. $\text{Trig}(x, y) \leftarrow \text{Unit}(x, c)\,,\ \text{Earthquake}(c, 1), \text{Result}_2^{\mathsf{Flip}}(0.6, \mathsf{Trig}, y, x)$
6a. $\exists y \text{Result}_2^{\mathsf{Flip}}(0.9, \mathsf{Trig}, x, y) \leftarrow \text{Burglary}(x, c, 1)$
6b. $\text{Trig}(x, y) \leftarrow \text{Burglary}(x, c, 1), \text{Result}_2^{\mathsf{Flip}}(0.9, \mathsf{Trig}, x, y)$
7. $\text{Alarm}(x) \leftarrow \text{Trig}(x, 1)$

Figure 5 The Datalog$^{\exists}$ program $\widehat{\mathcal{G}}$ for the GDatalog[Δ] program $\mathcal{G}$ of Figure 2.

other relation symbols of $\mathcal{I}^{\Delta}$ (namely, those of $\mathcal{I}$) are referred to as the *ordinary* relation symbols. Intuitively, a fact in Result_n^{δ} represents the result of a particular sample drawn from δ (where k is the number of parameters of δ and n is the number of optional arguments that form the event signature).

The set Θ^{Δ} contains all Datalog rules from Θ that have no Δ-terms. In addition, for every rule of the form $\psi(\mathbf{x}) \leftarrow \phi(\mathbf{x})$ in Θ, where ψ contains a Δ-term of the form $\delta[\![\mathbf{p}; \mathbf{q}]\!]$ with $n = |\mathbf{q}|$, Θ^{Δ} contains the rules $\exists y \text{Result}_n^{\delta}(\mathbf{p}, \mathbf{q}, y) \leftarrow \phi(\mathbf{x})$ and $\psi'(\mathbf{x}, y) \leftarrow \phi(\mathbf{x}), \text{Result}_n^{\delta}(\mathbf{p}, \mathbf{q}, y)$, where ψ' is obtained from ψ by replacing $\delta[\![\mathbf{p}; \mathbf{q}]\!]$ by y.

A *possible outcome* is defined as follows.

▶ **Definition 4** (Possible Outcome)**.** Let I be an input instance for a GDatalog[Δ] program $\mathcal{G}$. A *possible outcome* for I w.r.t. $\mathcal{G}$ is a minimal solution F of I w.r.t. $\widehat{\mathcal{G}}$, such that $\delta(b|\mathbf{p}) > 0$ for every distributional fact $\text{Result}_n^{\delta}(\mathbf{p}, \mathbf{q}, b) \in F$.
We denote the set of all possible outcomes of I w.r.t. $\mathcal{G}$ by $\Omega_{\mathcal{G}}(I)$, and we denote the set of all finite possible outcomes by $\Omega_{\mathcal{G}}^{\mathsf{fin}}(I)$.

▶ **Example 5.** The GDatalog[Δ] program $\mathcal{G}$ given in Example 2 gives rise to the Datalog$^{\exists}$ program $\widehat{\mathcal{G}}$ of Figure 5. For instance, Rule 6 of Figure 2 is replaced with Rules 6a and 6b of Figure 5. An example of a possible outcome for the input instance I is the instance consisting of the relations in Figure 6 (ignoring the "pr(f)" columns for now), together with the relations of I itself.

3.3 Probabilistic Semantics

The semantics of a GDatalog[Δ] program is a function that maps every input instance I to a probability distribution over $\Omega_{\mathcal{G}}(I)$. We now make this precise. For a distributional fact f of the form $\text{Result}_n^{\delta}(\mathbf{p}, \mathbf{q}, a)$, the *probability* of f, denoted $\text{pr}(f)$, is defined to be $\delta(a|\mathbf{p})$. For an ordinary (non-distributional) fact f, we define $\text{pr}(f) = 1$. For a finite set F of facts, we denote by $\mathbf{P}(F)$ the product of the probabilities of all the facts in F:[3]

$$\mathbf{P}(F) \stackrel{\text{def}}{=} \prod_{f \in F} \text{pr}(f)$$

[3] The product reflects the law of total probability and does *not* assume that different random choices are independent (and indeed, correlation is clear in the examples throughout the paper).

$\text{Result}_2^{\text{Flip}}$

p	att_1	att_2	$result$	$\text{pr}(f)$
0.01	Earthquake	Napa	1	0.01
0.01	Earthquake	Yucaipa	0	0.99
0.9	Trig	NP1	1	0.9
0.9	Trig	NP3	0	0.1
0.6	Trig	NP1	1	0.6
0.6	Trig	NP2	1	0.6
0.6	Trig	NP3	0	0.4

Unit

id	$city$
NP1	Napa
NP2	Napa
NP3	Napa
YC1	Yucaipa

Earthquake

$city$	eq
Napa	1
Yucaipa	0

Alarm

$unit$
NP1
NP2

$\text{Result}_3^{\text{Flip}}$

p	att_1	att_2	att_3	$result$	$\text{pr}(f)$
0.03	Burglary	NP1	Napa	1	0.03
0.03	Burglary	NP2	Napa	0	0.97
0.03	Burglary	NP3	Napa	1	0.03
0.01	Burglary	YC1	Yucaipa	0	0.99

Burglary

$unit$	$city$	$draw$
NP1	Napa	1
NP2	Napa	0
NP3	Napa	1
YC1	Yucaipa	0

Trig

$unit$	$Trig$
NP1	1
NP3	0
NP2	1
NP3	0

Figure 6 A possible outcome for the input instance I in the burglar example.

▶ **Example 6** (continued). Let J be the instance that consists of all of the relations in Figures 1 and 6. As we already remarked, J is a possible outcome of I w.r.t. $\mathcal{G}$. For convenience, in the case of distributional relations, we have indicated the probability of each fact next to the corresponding row. $\mathbf{P}(J)$ is the product of all of the numbers in the columns titled "$\text{pr}(f)$," that is, $0.01 \times 0.99 \times 0.9 \times \cdots \times 0.99$.

One can easily come up with examples where possible outcomes are infinite, and in fact, the space $\Omega_{\mathcal{G}}(I)$ of all possible outcomes is uncountable. Hence, we need to consider probability spaces over uncountable domains; those are defined by means of measure spaces.

Let $\mathcal{G}$ be a GDatalog[Δ] program, and let I be an input for $\mathcal{G}$. We say that a finite sequence $\mathbf{f} = (f_1, \ldots, f_n)$ of facts is a *derivation* (w.r.t. I) if for all $i = 1, \ldots, n$, the fact f_i is the result of applying some rule of $\mathcal{G}$ that is not satisfied in $I \cup \{f_1, \ldots, f_{i-1}\}$ (in the case of applying a rule with a Δ-atom in the head, choosing a value randomly). If $f_1, \ldots, f_n$ is a derivation, then the set $\{f_1, \ldots, f_n\}$ is a *derivation set*. Hence, a finite set F of facts is a derivation set if and only if $I \cup F$ is an intermediate instance in some chase tree.

Let $\mathcal{G}$ be a GDatalog[Δ] program, I be an input for $\mathcal{G}$, and F be a set of facts. We denote by $\Omega_{\mathcal{G}}^{F\subseteq}(I)$ the set of all possible outcomes $J \subseteq \Omega_{\mathcal{G}}(I)$ such that $F \subseteq J$. The following theorem states how we determine the probability space defined by a GDatalog[Δ] program.

▶ **Theorem 7.** *Let $\mathcal{G}$ be a GDatalog[Δ] program, and let I be an input for $\mathcal{G}$. There exists a* unique *probability measure space $(\Omega, \mathcal{F}, \pi)$, denoted $\mu_{\mathcal{G},I}$, that satisfies all of the following.*

1. $\Omega = \Omega_{\mathcal{G}}(I)$;
2. *$(\Omega, \mathcal{F})$ is the σ-algebra generated from the sets of the form $\Omega_{\mathcal{G}}^{F\subseteq}(I)$ where F is finite;*
3. *$\pi(\Omega_{\mathcal{G}}^{F\subseteq}(I)) = \mathbf{P}(F)$ for every derivation set F.*

Moreover, if J is a finite possible outcome, then $\pi(\{J\})$ is equal to $\mathbf{P}(J)$.

Theorem 7 provides us with a semantics for GDatalog[Δ] programs: the semantics of a GDatalog[Δ] program $\mathcal{G}$ is a map from input instances I to probability measure spaces $\mu_{\mathcal{G},I}$ over possible outcomes (as uniquely determined by Theorem 7). The proof of Theorem 7 is by means of the *chase procedure*, which we discuss in the next section. A direct corollary of the theorem applies to the important case where all possible outcomes are finite (and the probability space may be infinite, but necessarily discrete).

▶ **Corollary 8.** *Let $\mathcal{G}$ be a GDatalog[Δ] program, and I an input instance for $\mathcal{G}$, such that $\Omega_{\mathcal{G}}(I) = \Omega^{\mathsf{fin}}_{\mathcal{G}}(I)$. Then $\mathbf{P}$ is a discrete probability function over $\Omega_{\mathcal{G}}(I)$; that is, $\sum_{J \in \Omega_{\mathcal{G}}(I)} \mathbf{P}(J) = 1$.*

3.4 Finiteness and Weak Acyclicity

Corollary 8 applies only when all solutions are finite, that is, $\Omega_{\mathcal{G}}(I) = \Omega^{\mathsf{fin}}_{\mathcal{G}}(I)$. We now present the notion of *weak acyclicity* for a GDatalog[Δ] program, as a natural syntactic condition that guarantees finiteness of all possible outcomes (for all input instances). This draws on the notion of weak acyclicity for Datalog$^{\exists}$ [18]. Consider any GDatalog[Δ] program $\mathcal{G} = (\mathcal{E}, \mathcal{I}, \Theta)$. A *position* of $\mathcal{I}$ is a pair (R, i) where $R \in \mathcal{I}$ and i is an attribute of R. The *dependency graph* of $\mathcal{G}$ is the directed graph that has the positions of $\mathcal{I}$ as the nodes, and the following edges:

- A *normal edge* $(R, i) \to (S, j)$ whenever there is a rule $\psi(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$ and a variable x occurring at position (R, i) in $\varphi(\mathbf{x})$, and at position (S, j) in $\psi(\mathbf{x})$.
- A *special edge* $(R, i) \to^* (S, j)$ whenever there is a rule of the form

$$S(t_1, \ldots, t_{j-1}, \delta[\![\mathbf{p}; \mathbf{q}]\!], t_{j+1}, \ldots, t_n) \leftarrow \varphi(\mathbf{x})$$

  and a variable x occurring at position (R, i) in $\varphi(\mathbf{x})$ as well as in $\mathbf{p}$ or $\mathbf{q}$.

We say that $\mathcal{G}$ is *weakly acyclic* if no cycle in its dependency graph contains a special edge.

▶ **Theorem 9.** *If a GDatalog[Δ] program $\mathcal{G}$ is weakly acyclic, then $\Omega_{\mathcal{G}}(I) = \Omega^{\mathsf{fin}}_{\mathcal{G}}(I)$ for all input instances I.*

▶ **Example 10.** The burglar example program in Figure 2 is easily seen to be weakly acyclic (indeed, every non-recursive GDatalog[Δ] program is weakly-acyclic). In the case of the virus-dissemination example, the dependency graph in Figure 3 shows that, although this program features recursion, it is weakly acyclic as well.

3.5 Discussion

We conclude this section with some comments. First, we note that the restriction of a conclusion of a rule to include a single Δ-term significantly simplifies the presentation, but does not reduce the expressive power. In particular, we could simulate multiple Δ-terms in the conclusion using a collection of predicates and rules. For example, if one wishes to have conclusion where a person gets both a random height and a random weight (possibly with shared parameters), then she can do so by deriving $\mathrm{PersonHeight}(p, h)$ and $\mathrm{PersonWeight}(p, w)$ separately, and using the rule $\mathrm{PersonHW}(p, h, w) \leftarrow \mathrm{PersonHeight}(p, h), \mathrm{PersonWeight}(p, w)$. We also highlight the fact that our framework can easily simulate the probabilistic database model of *independent tuples* [46] with probabilities mentioned in the database. The framework can also simulate Bayesian networks, given relations that store the conditional probability tables, using the appropriate numerical distributions (e.g., Flip for the case of Boolean random variables). In addition, we note that a *disjunctive* Datalog rule [16], where the conclusion can be a disjunction of atoms, can be simulated by our model (with probabilities ignored): If the conclusion has n disjuncts, then we construct a distributional rule with a probability distribution over $\{1, \ldots, n\}$, and additional n deterministic rules corresponding to the atoms.

4 Chasing Generative Programs

The chase [29, 3] is a classic technique used for reasoning about database integrity constraints such as *tuple-generating dependencies*. This technique can be equivalently viewed as a

tableaux-style proof system for $\forall^*\exists^*$-Horn sentences. In the special case of *full* tuple-generating dependencies, which are syntactically isomorphic to Datalog rules, the chase is closely related to (a tuple-at-a-time version of) the naive *bottom-up evaluation* strategy for Datalog program (cf. [2]). We now present a suitable variant of the chase for generative Datalog programs, and use it in order to construct the probability space of Theorem 7.

We note that, although the notions and results could arguably be phrased in terms of a probabilistic extension of the bottom-up Datalog evaluation strategy, the fact that a GDatalog[Δ] rule can create new values makes it more convenient to phrase them in terms of a suitable adaptation of the chase procedure.

Throughout this section, we fix a GDatalog[Δ] program $\mathcal{G} = (\mathcal{E}, \mathcal{I}, \Theta)$ and its associated Datalog$^\exists$ program $\widehat{\mathcal{G}} = (\mathcal{E}, \mathcal{I}^\Delta, \Theta^\Delta)$. We first define the notions of *chase step* and *chase tree*.

Chase step. Consider an instance J, a rule $\tau \in \Theta^\Delta$ of the form $\psi(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$, and a tuple $\mathbf{a}$ such that $\varphi(\mathbf{a})$ is satisfied in J but $\psi(\mathbf{a})$ is not satisfied in J. If $\psi(\mathbf{x})$ is a distributional atom of the form $\exists y \mathrm{Result}_i^\delta(\mathbf{p}, \mathbf{q}, y)$, then ψ being "not satisfied" is interpreted in the logical sense (regardless of probabilities): there is no y such that $(\mathbf{p}, \mathbf{q}, y)$ is in Result_i^δ. In that case, let $\mathcal{J}$ be the set of all instances J_b obtained by extending J with $\psi(\mathbf{a})$ for a specific value b of the existential variable y, such that $\delta(b|\mathbf{p}) > 0$. Furthermore, let π be the discrete probability distribution over $\mathcal{J}$ that assigns to J_b the probability $\delta(b|\mathbf{p})$. If $\psi(\mathbf{x})$ is an ordinary atom without existential quantifiers, $\mathcal{J}$ is simply defined as $\{J'\}$, where J' extends J with the fact $\psi(\mathbf{a})$, and $\pi(J') = 1$. We say that $J \xrightarrow{\tau(\mathbf{a})} (\mathcal{J}, \pi)$ is a *valid chase step*.

Chase tree. Let I be an input instance for $\mathcal{G}$. A *chase tree for* I w.r.t. $\mathcal{G}$ is a possibly infinite tree, whose nodes are labeled by instances over $\mathcal{E} \cup \mathcal{I}$, and whose edges are labeled by real numbers, such that:

1. The root is labeled by I;
2. For each non-leaf node labeled J, if $\mathcal{J}$ is the set of labels of the children of the node, and if π is the map assigning to each $J' \in \mathcal{J}$ the label of the edge from J to J', then $J \xrightarrow{\tau(\mathbf{a})} (\mathcal{J}, \pi)$ is a valid chase step for some rule $\tau \in \Theta^\Delta$ and tuple $\mathbf{a}$.
3. For each leaf node labeled J, there does not exist a valid chase step of the form $J \xrightarrow{\tau(\mathbf{a})} (\mathcal{J}, \pi)$. In other words, the tree cannot be extended to a larger chase tree.

We denote by $L(v)$ the label (instance) of the node v. Each $L(v)$ is said to be an *intermediate instance* w.r.t. the chase tree. Consider a GDatalog[Δ] program $\mathcal{G}$ and an input I for $\mathcal{G}$. A *maximal path* of a chase tree T is a path P that starts with the root, and either ends in a leaf or is infinite. Observe that the labels (instances) along a maximal path form a chain (w.r.t. the set-containment partial order). A maximal path P of a chase tree is *fair* if whenever the premise of a rule is satisfied by some tuple in some intermediate instance on P, then the conclusion of the rule is satisfied for the same tuple in some intermediate instance on P. A chase tree T is *fair* (or has the *fairness* property) if every maximal path is fair. Note that finite chase trees are fair. We restrict attention to fair chase trees. Fairness is a classic notion in the study of infinite computations;[4] moreover, fair chase trees can be constructed, for example, by maintaining a queue of "active rule firings."

Let $\mathcal{G}$ be a GDatalog[Δ] program, I be an input for $\mathcal{G}$, and T be a chase tree. We denote by $\mathit{paths}(T)$ the set of maximal paths of T. (Note that $\mathit{paths}(T)$ may be uncountably infinite.) For $P \in \mathit{paths}(T)$, we denote by $\cup P$ the union of the (chain of) labels $L(v)$ along P.

[4] Cf. any textbook on term rewriting systems or lambda calculus.

▶ **Theorem 11.** *Let $\mathcal{G}$ be a GDatalog[Δ] program, I an input for $\mathcal{G}$, and T a fair chase tree. The mapping $P \to \cup P$ is a bijection between $paths(T)$ and $\Omega_{\mathcal{G}}(I)$.*

Chase measure. Let $\mathcal{G}$ be a GDatalog[Δ] program, let I be an input for $\mathcal{G}$, and let T be a chase tree. Our goal is to define a probability measure over $\Omega_{\mathcal{G}}(I)$. Given Theorem 11, we can do that by defining a probability measure over $paths(T)$. A random path in $paths(T)$ can be viewed as a *Markov chain* that is defined by a random walk over T, starting from the root. A measure space for such a Markov chain is defined by means of *cylindrification* [7]. Let v be a node of T. The *v-cylinder* of T, denoted C_v^T, is the subset of $paths(T)$ that consists of all the maximal paths that contain v. A *cylinder* of T is a subset of $paths(T)$ that forms a v-cylinder for some node v. We denote by $C(T)$ the set of all the cylinders of T.

Recall that $L(v)$ is a finite set of facts, and observe that $\mathbf{P}(L(v))$ is the product of the probabilities along the path from the root to v. The following theorem is a special case of a classic result on Markov chains (cf. [7]).

▶ **Theorem 12.** *Let $\mathcal{G}$ be a GDatalog[Δ] program, let I be an input for $\mathcal{G}$, and let T be a chase tree. There exists a unique probability measure $(\Omega, \mathcal{F}, \pi)$ that satisfies all of the following.*

1. $\Omega = paths(T)$.
2. $(\Omega, \mathcal{F})$ *is the σ-algebra generated from $C(T)$.*
3. $\pi(C_v^T) = \mathbf{P}(L(v))$ *for all nodes v of T.*

Theorems 11 and 12 suggest the following definition.

▶ **Definition 13** (Chase Probability Measure)**.** Let $\mathcal{G}$ be a GDatalog[Δ] program, let I be an input for $\mathcal{G}$, let T be a chase tree, and let $(\Omega, \mathcal{F}, \pi)$ be the probability measure of Theorem 12. The probability measure μ_T over $\Omega_{\mathcal{G}}(I)$ is the one obtained from $(\Omega, \mathcal{F}, \pi)$ by replacing every maximal path P with the possible outcome $\cup P$.

The following theorem states that the probability measure space represented by a chase tree is independent of the specific chase tree of choice.

▶ **Theorem 14.** *Let $\mathcal{G}$ be a GDatalog[Δ] program, let I be an input for $\mathcal{G}$, and let T and T' be two fair chase trees. Then $\mu_T = \mu_{T'}$.*

5 Probabilistic-Programming Datalog

To complete our framework, we define *probabilistic-programming Datalog*, *PPDL* for short, wherein a program augments a generative Datalog program with constraints; these constraints unify the traditional *integrity constraints* of databases and the traditional *observations* of probabilistic programming.

▶ **Definition 15** (PPDL[Δ])**.** Let Δ be a finite set of parameterized numerical distributions. A *PPDL*[Δ] *program* is a quadruple $(\mathcal{E}, \mathcal{I}, \Theta, \Phi)$, where $(\mathcal{E}, \mathcal{I}, \Theta)$ is a GDatalog[Δ] program and Φ is a finite set of logical constraints over $\mathcal{E} \cup \mathcal{I}$.[5]

▶ **Example 16.** Consider again Example 2. Suppose that we have the EDB relations ObservedHAlarm and ObservedBAlarm that represent observed home and business alarms, respectively. We obtain from the program in the example a PPDL[Δ]-program by adding the following constraints:

[5] We will address the choice of constraint language, and its algorithmic impact, in future work.

1. $\text{ObservedHAlarm}(h) \rightarrow \text{Alarm}(h)$
2. $\text{ObservedBAlarm}(b) \rightarrow \text{Alarm}(b)$

We use right (in contrast to left) arrows to distinguish constraints from ordinary Datalog rules. Note that a possible outcome J of an input instance I satisfies these constraints if J contains $\text{Alarm}(x)$ for all $x \in \text{ObservedHAlarm}^I \cup \text{ObservedBAlarm}^I$.

A PPDL[Δ] program defines the posterior distribution over its GDatalog[Δ] program, conditioned on the satisfaction of the constraints. A formal definition follows.

Let $\mathcal{P} = (\mathcal{E}, \mathcal{I}, \Theta, \Phi)$ be a PPDL[Δ] program, and let $\mathcal{G}$ be the GDatalog[Δ] program $(\mathcal{E}, \mathcal{I}, \Theta)$. An *input instance* for $\mathcal{P}$ is an input instance I for $\mathcal{G}$. We say that I is a *legal* input instance if $\{J \in \Omega_{\mathcal{G}}(I) \mid J \models \Phi\}$ is a measurable set in the probability space $\mu_{\mathcal{G},I}$, and its measure is nonzero. Intuitively, I is legal if it is consistent with the observations (i.e., with the constraints in Φ), given $\mathcal{G}$. The semantics of a PPDL[Δ] program is defined as follows.

▶ **Definition 17.** Let $\mathcal{P} = (\mathcal{E}, \mathcal{I}, \Theta, \Phi)$ be a PPDL[Δ] program, $\mathcal{G}$ the GDatalog[Δ] program $(\mathcal{E}, \mathcal{I}, \Theta)$, I a legal input instance for $\mathcal{P}$, and $\mu_{\mathcal{G},I} = (\Omega_{\mathcal{G}}(I), \mathcal{F}_{\mathcal{G}}, \pi_{\mathcal{G}})$. The probability space defined by $\mathcal{P}$ and I, denoted $\mu_{\mathcal{P},I}$, is the triple $(\Omega_{\mathcal{P}}(I), \mathcal{F}_{\mathcal{P}}, \pi_{\mathcal{P}})$ where:

1. $\Omega_{\mathcal{P}}(I) = \{J \in \Omega_{\mathcal{G}}(I) \mid J \models \Phi\}$
2. $\mathcal{F}_{\mathcal{P}} = \{S \cap \Omega_{\mathcal{P}}(I) \mid S \in \mathcal{F}_{\mathcal{G}}\}$
3. $\pi_{\mathcal{P}}(S) = \pi_{\mathcal{G}}(S)/\pi_{\mathcal{G}}(\Omega_{\mathcal{P}}(I))$ for every $S \in \mathcal{F}_{\mathcal{P}}$.

In other words, $\mu_{\mathcal{P},I}$ is $\mu_{\mathcal{G},I}$ conditioned on Φ.

▶ **Example 18.** Continuing Example 16, the semantics of this program is the posterior probability distribution that is obtained from the prior of Example 2, by conditioning on the fact that $\text{Alarm}(x)$ holds for all $x \in \text{ObservedHAlarm}^I \cup \text{ObservedBAlarm}^I$. Similarly, using an additional constraint we can express the condition that an alarm is off unless observed. One can ask various natural queries over this probability space of possible outcomes, such as the probability of the fact $\text{Earthquake}(\mathsf{Napa}, 1)$.

We note that when G is weakly acyclic, the event defined by Φ is measurable (since in that case the probability space is discrete) and the definition of legality boils down to the existence of a possible outcome.

5.1 Invariance under First-Order Equivalence

PPDL[Δ] programs are fully declarative in a strong sense: syntactically their rules and constraints can be viewed as first-order theories. Moreover, whenever two PPDL[Δ] programs, viewed in this way, are logically equivalent, then they are equivalent as PPDL[Δ] programs, in the sense that they give rise to the same set of possible outcomes and the same probability distribution over possible outcomes.

Formally, we say that two PPDL[Δ] programs, $\mathcal{P}_1 = (\mathcal{E}, \mathcal{I}, \Theta_1, \Phi_1)$ and $\mathcal{P}_2 = (\mathcal{E}, \mathcal{I}, \Theta_2, \Phi_2)$, are *semantically equivalent* if, for all input instances I, the probability spaces $\mu_{\mathcal{P}_1,I}$ and $\mu_{\mathcal{P}_2,I}$ coincide. Syntactically, the rules and constraints of a PPDL[Δ] program can be viewed as a finite first-order theory over a signature consisting of relation symbols, constant symbols, and function symbols (here, if the same name of a function name is used with different numbers of arguments, such as Flip in Figure 2, we treat them as distinct function symbols). We say that $\mathcal{P}_1$ and $\mathcal{P}_2$ are *FO-equivalent* if, viewed as first-order theories, Θ_1 is logically equivalent to Θ_2 (i.e., the two theories have the same models) and likewise for Φ_1 and Φ_2. We have the following theorems.

▶ **Theorem 19.** *If two PPDL[Δ] programs are FO-equivalent, then they are semantically equivalent (but not necessarily vice versa).*

▶ **Theorem 20.** *First-order equivalence is decidable for weakly acyclic GDatalog*[Δ] *programs. Semantic equivalence is undecidable for weakly acyclic GDatalog*[Δ] *programs (in fact, even for* $\Delta = \emptyset$*).*

6 Related Work

Our contribution is a marriage between probabilistic programming and the declarative specification of Datalog. The key features of our approach are the ability to express probabilistic models *concisely* and *declaratively* in a Datalog extension with probability distributions as first-class citizens. Existing formalisms that associate a probabilistic interpretation with logic are either not declarative (at least in the Datalog sense) or depart from the probabilistic programming paradigm (e.g., by lacking the support for numerical probability distributions). We next discuss representative related formalisms and contrast them with our work. They can be classified into three broad categories: *(1)* imperative specifications over logical structures, *(2)* logic over probabilistic databases, *and (3)* indirect specifications over the Herbrand base. (Some of these formalisms belong to more than one category.)

The first category includes imperative probabilistic programming languages [42]. We also include in this category declarative specifications of Bayesian networks, such as BLOG [32] and P-log [8]. Although declarative in nature, these languages inherently assume a form of acyclicity that allows the rules to be executed serially. Here we are able to avoid such an assumption since our approach is based on the minimal solutions of an existential Datalog program. The program in Figure 3, for example, uses recursion (as is typically the case for probabilistic models in social network analysis). In particular, it is not clear how this program can be phrased by translation into a Bayesian network. BLOG can express probability distributions over logical structures, via generative stochastic models that can draw values at random from numerical distributions, and condition values of program variables on observations. In contrast with closed-universe languages such as SQL and logic programs, BLOG considers open-universe probability models that allow for uncertainty about the existence and identity of objects.

The formalisms in the second category view the generative part of the specification of a statistical model as a two-step process. In the first step, facts are randomly generated by a mechanism external to the program. In the second step, a logic program, such as Prolog [26] or Datalog [1], is evaluated over the resulting random structure. This approach has been taken by PRISM [44], the *Independent Choice Logic* [41], and to a large extent by *probabilistic databases* [46] and their semistructured counterparts [25]. We focus on a formalism that completely defines the statistical model, without referring to external processes. As an important example, in PPDL one can sample from distributions that have parameters that by themselves are randomly generated using the program. This is the common practice in Bayesian machine learning (e.g., logistic regression), but it is not clear how it can be done within approaches of the second category.

One step beyond the second category and closer to our work is taken by uncertainty-aware query languages for probabilistic data such as TriQL [48], I-SQL, and world-set algebra [4, 5]. The latter two are natural analogs to SQL and relational algebra for the case of incomplete information and probabilistic data [4]. They feature constructs such as `repair-key`, `choice-of`, `possible`, and `group-worlds-by` that can construct possible worlds representing all repairs of a relation w.r.t. key constraints, close the possible worlds by unioning or intersecting them, or group the worlds into sets with the same results to sub-queries. World-set algebra has been extended to (world-set) Datalog, fixpoint, and

while-languages [14] to define Markov chains. While such languages cannot explicitly specify probability distributions, they may simulate a specific categorical distribution indirectly using non-trivial programs with specialized language constructs like `repair-key` on input tuples with weights representing samples from the distribution.

MCDB [24] and SimSQL [12] propose SQL extensions (with for-loops and probability distributions) coupled with Monte Carlo simulations and parallel database techniques for stochastic analytics in the database. Their formalism does not involve the semantic challenges that we have faced in this paper. Although being based on SQL, these extensions do not offer a truly declarative means to specify probabilistic models, and end up being more similar to the imperative languages mentioned under the first category.

Formalisms in the third category use rule weighting as indirect specifications of probability spaces over the *Herbrand base*, which is the set of all the facts that can be obtained using the predicate symbols and the constants of the database. This category includes *Markov Logic Networks (MLNs)* [15, 34], where the logical rules are used as a compact and intuitive way of defining *factors*. In other words, the probability of a possible world is the product of all the numbers (factors) that are associated with the grounded rules that the world satisfies. This approach is applied in DeepDive [35], where a database is used for storing relational data and extracted text, and database queries are used for defining the factors of a factor graph. We view this approach as *indirect* since a rule does not determine directly the distribution of values. Moreover, the semantics of rules is such that the addition of a rule that is logically equivalent to (or implied by, or indeed equal to) an existing rule changes the semantics and thus the probability distribution. A similar approach is taken by *Probabilistic Soft Logic* [11], where in each possible world every fact is associated with a degree of truth.

Further formalisms in this category are *probabilistic Datalog* [19], *probabilistic Datalog+/-* [21], and *probabilistic logic programming (ProbLog)* [26]. There, every rule is associated with a probability. For ProbLog, the semantics is not declarative as the rules follow a certain evaluation order; for probabilistic Datalog, the semantics is purely declarative. Both semantics are different from ours and that of the other formalisms mentioned thus far. A Datalog rule is interpreted as a rule over a probability distribution over possible worlds, and it states that, for a given grounding of the rule, the marginal probability of being true is as stated in the rule. Probabilistic Datalog+/- uses MLNs as the underlying semantics. Besides our support for numerical probability distributions, our formalism is used for defining a single probability space, which is in par with the standard practice in probabilistic programming.

As discussed earlier, GDatalog[Δ] allows for recursion, and the semantics is captured by (possibly infinite) Markov chains. Related formalisms are that of *Probabilistic Context-Free Grammars* (PCFG) and the more general *Recursive Markov Chains* (RMC) [17], where the probabilistic specification is by means of a finite set of transition graphs that can call one another (in the sense of method calls) in a possibly recursive fashion. In the database literature, PCFGs and RMCs are used in the context of probabilistic XML [13, 9]. These formalisms do not involve numerical distributions. In future work, we plan to study their relative expressive power compared to restrictions of our framework.

7 Concluding Remarks

We proposed and investigated a declarative framework for specifying statistical models in the context of a database, based on a conservative extension of Datalog with numerical distributions. The framework differs from existing probabilistic programming languages not only due to the tight integration with a database, but also because of its fully declarative

rule-based language: the interpretation of a program is independent of transformations (such as reordering or duplication of rules) that preserve the first-order semantics. This was achieved by treating a GDatalog[Δ] program as a Datalog program with existentially quantified variables in the conclusion of rules, and applying a suitable variant of the chase.

This paper opens various important directions for future work. One direction is to establish tractable conditions that guarantee that a given input is legal. Also, an interesting problem is to detect conditions under which the chase is a *self conjugate* [43], that is, the probability space $\mu_{\mathcal{P},I}$ is captured by a chase procedure without backtracking.

Our ultimate goal is to develop a full-fledged PP system based on the declarative specification language that we proposed here. In this work we focused on the foundations and robustness of the specification language. As in other PP languages, inference, such as computing the marginal probability of an IDB fact, is a challenging aspect, and we plan to investigate the application of common approaches such as sampling-based and lifted-inference techniques. We believe that the declarative nature of PPDL can lead to identifying interesting fragments that admit tractable complexity due to specialized techniques, just as is the case for Datalog evaluation in databases.

Practical applications will require further extensions to the language. We plan to support continuous probability distributions (e.g., continuous uniform, Pareto, and Gaussian), which are often used in statistical models. Syntactically, this extension is straightforward: we just need to include these distributions in Δ. Likewise, extending the probabilistic chase is also straightforward. More challenging is the semantic analysis, and, in particular, the definition of the probability space induced by the chase. We also plan to extend PPDL to support distributions that take a variable (and unbounded) number of parameters. A simple example is the *categorical* distribution where a single member of a finite domain of items is to be selected, each item with its own probability; in this case we can adopt the `repair-key` operation of the world-set algebra [4, 5]. Finally, we plan to add support for *multivariate distributions*, which are distributions with a support in $\mathbb{R}^k$ for $k > 1$ (where, again, k can be variable and unbounded). Examples of popular such distributions are multinomial, Dirichlet, and multivariate Gaussian distribution.

At LogicBlox, we are working on extending LogiQL with PPDL. An interesting syntactic and semantic challenge is that a program should contain rules of two kinds: probabilistic programming (i.e., PPDL rules) and inference over probabilistic programs (e.g., find the most likely execution). The latter rules involve the major challenge of efficient inference over PPDL. Towards that, our efforts fall in three different directions. First, we implement samplers of random executions. Second, we translate programs of restricted fragments into external statistical solvers (e.g., Bayesian Network libraries and *sequential Monte Carlo* [45]). Third, we are looking into fragments where we can apply exact and efficient (lifted) inference.

Acknowledgments. We are thankful to Molham Aref, Todd J. Green and Emir Pasalic for insightful discussions and feedback on this work. We also thank Michael Benedikt, Georg Gottlob and Yannis Kassios for providing useful comments and suggestions. Benny Kimelfeld is a Taub Fellow, supported by the Taub Foundation. Kimelfeld's work was supported in part by the Israel Science Foundation. Dan Olteanu acknowledges the support of the EPSRC programme grant VADA. This work was supported by DARPA under agreements #FA8750-15-2-0009 and #FA8750-14-2-0217. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright thereon.

References

1 Serge Abiteboul, Daniel Deutch, and Victor Vianu. Deduction with contradictions in Datalog. In *ICDT*, pages 143–154, 2014.

2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

3 Alfred V. Aho, Catriel Beeri, and Jeffrey D. Ullman. The theory of joins in relational databases. *ACM Trans. on Datab. Syst.*, 4(3):297–314, 1979.

4 Lyublena Antova, Christoph Koch, and Dan Olteanu. From complete to incomplete information and back. In *SIGMOD*, pages 713–724, 2007. doi:10.1145/1247480.1247559.

5 Lyublena Antova, Christoph Koch, and Dan Olteanu. Query language support for incomplete information in the MayBMS system. In *VLDB*, pages 1422–1425, 2007.

6 Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and implementation of the LogicBlox system. In *PDOS*, pages 1371–1382. ACM, 2015.

7 Robert B. Ash and Catherine Doleans-Dade. *Probability & Measure Theory*. Harcourt Academic Press, 2000.

8 Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. *Theory Pract. Log. Program.*, 9(1):57–144, 2009.

9 Michael Benedikt, Evgeny Kharlamov, Dan Olteanu, and Pierre Senellart. Probabilistic XML via Markov chains. *PVLDB*, 3(1):770–781, 2010.

10 David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. of Machine Learning Research*, 3:993–1022, 2003.

11 Matthias Bröcheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *UAI*, pages 73–82, 2010.

12 Zhuhua Cai, Zografoula Vagena, Luis Leopoldo Perez, Subramanian Arumugam, Peter J. Haas, and Christopher M. Jermaine. Simulation of database-valued Markov chains using SimSQL. In *SIGMOD*, pages 637–648, 2013.

13 Sara Cohen and Benny Kimelfeld. Querying parse trees of stochastic context-free grammars. In *ICDT*, pages 62–75. ACM, 2010.

14 Daniel Deutch, Christoph Koch, and Tova Milo. On probabilistic fixpoint and Markov chain query languages. In *PODS*, pages 215–226, 2010. doi:10.1145/1807085.1807114.

15 Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on AI and Machine Learning. Morgan & Claypool Publishers, 2009.

16 Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

17 Kousha Etessami and Mihalis Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1), 2009. doi:10.1145/1462153.1462154.

18 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

19 Norbert Fuhr. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *JASIS*, 51(2):95–110, 2000.

20 Noah D. Goodman. The principles and practice of probabilistic programming. In *POPL*, pages 399–402, 2013.

21 Georg Gottlob, Thomas Lukasiewicz, MariaVanina Martinez, and Gerardo Simari. Query answering under probabilistic uncertainty in Datalog+/- ontologies. *Annals of Math.& AI*, 69(1):37–72, 2013. doi:10.1007/s10472-013-9342-1.

22 Terry Halpin and Spencer Rugaber. *LogiQL: A Query Language for Smart Databases*. CRC Press, 2014.

23 Shan Shan Huang, Todd Jeffrey Green, and Boon Thau Loo. Datalog and emerging applications: An interactive tutorial. In *SIGMOD*, pages 1213–1216, 2011. doi:10.1145/1989323.1989456.

24 Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine, and Peter J. Haas. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.

25 Benny Kimelfeld and Pierre Senellart. Probabilistic XML: models and complexity. In *Advances in Probabilistic Databases for Uncertain Information Management*, volume 304 of *Studies in Fuzziness and Soft Computing*, pages 39–66. Springer, 2013.

26 Angelika Kimmig, Bart Demoen, Luc De Raedt, Vitor Santos Costa, and Ricardo Rocha. On the implementation of the probabilistic logic programming language ProbLog. *Theory and Practice of Logic Programming*, 11:235–262, 2011. doi:10.1017/S1471068410000566.

27 Lyric Labs. Chimple. URL: http://chimple.probprog.org/.

28 Boon Thau Loo, Tyson Condie, Minos N. Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, 2009. doi:10.1145/1592761.1592785.

29 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. on Datab. Syst.*, 4(4):455–469, 1979.

30 Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099, 2014. URL: http://arxiv.org/abs/1404.0099.

31 Andrew Kachites McCallum. Multi-label text classification with a mixture model trained by EM. In *Assoc. for the Advancement of Artificial Intelligence workshop on text learning*, 1999.

32 B. Milch and et al. BLOG: Probabilistic models with unknown objects. In *IJCAI*, pages 1352–1359, 2005.

33 Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, pages 103–134, 2000.

34 Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up statistical inference in Markov Logic Networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.

35 Feng Niu, Ce Zhang, Christopher Re, and Jude W. Shavlik. DeepDive: Web-scale knowledge-base construction using statistical learning and inference. In *Int. Workshop on Searching and Integrating New Web Data Sources*, volume 884 of *CEUR Workshop Proceedings*, pages 25–28, 2012.

36 Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. R2: an efficient MCMC sampler for probabilistic programs. In *AAAI*, pages 2476–2482, 2014.

37 Brooks Paige and Frank Wood. A compilation target for probabilistic programming languages. In *ICML*, volume 32, pages 1935–1943, 2014.

38 Anand Patil, David Huard, and Christopher J. Fonnesbeck. PyMC: Bayesian Stochastic Modelling in Python. *J. of Statistical Software*, 35(4):1–81, 2010.

39 Judea Pearl. *Probabilistic reasoning in intelligent systems – networks of plausible inference*. Morgan Kaufmann, 1989.

40 Avi Pfeffer. Figaro: An object-oriented probabilistic programming language. Technical report, Charles River Analytics, 2009.

41 David Poole. The independent choice logic and beyond. In *Probabilistic Inductive Logic Programming – Theory and Applications*, pages 222–243, 2008.

42 Repository on probabilistic programming languages, 2014. URL: http://www.probabilistic-programming.org.

43 H. Raiffa and R. Schlaifer. *Applied Statistical Decision Theory*. Harvard University Press, Harvard, 1961.

44 Taisuke Sato and Yoshitaka Kameya. PRISM: A language for symbolic-statistical modeling. In *IJCAI*, pages 1330–1339, 1997.

45 Adrian Smith, Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo methods in practice*. Springer Science & Business Media, 2013.

46 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

47 Balder ten Cate, Benny Kimelfeld, and Dan Olteanu. PPDL: probabilistic programming with Datalog. In *AMW*, volume 1378 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

48 Jennifer Widom. Trio: a system for data, uncertainty, and lineage. In Charu Aggarwal, editor, *Managing and Mining Uncertain Data*, chapter 5. Springer-Verlag, 2008.

Worst-Case Optimal Algorithms for Parallel Query Processing*

Paraschos Koutris[1], Paul Beame[2], and Dan Suciu[3]

1 **University of Washington, Seattle, WA, USA**
pkoutris@cs.washington.edu
2 **University of Washington, Seattle, WA, USA**
beame@cs.washington.edu
3 **University of Washington, Seattle, WA, USA**
suciu@cs.washington.edu

Abstract

In this paper, we study the communication complexity for the problem of computing a conjunctive query on a large database in a parallel setting with p servers. In contrast to previous work, where upper and lower bounds on the communication were specified for particular structures of data (either data without skew, or data with specific types of skew), in this work we focus on *worst-case analysis* of the communication cost. The goal is to find worst-case optimal parallel algorithms, similar to the work of [17] for sequential algorithms.

We first show that for a single round we can obtain an optimal worst-case algorithm. The optimal load for a conjunctive query q when all relations have size equal to M is $O(M/p^{1/\psi^*})$, where ψ^* is a new query-related quantity called the *edge quasi-packing number*, which is different from both the edge packing number and edge cover number of the query hypergraph. For multiple rounds, we present algorithms that are optimal for several classes of queries. Finally, we show a surprising connection to the external memory model, which allows us to translate parallel algorithms to external memory algorithms. This technique allows us to recover (within a polylogarithmic factor) several recent results on the I/O complexity for computing join queries, and also obtain optimal algorithms for other classes of queries.

1998 ACM Subject Classification H.2.4 [Systems] Query Processing

Keywords and phrases conjunctive query, parallel computation, worst-case bounds

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.8

1 Introduction

The last decade has seen the development and widespread use of massively parallel systems that perform data analytics tasks over big data: examples of such systems are MapReduce [7], Dremel [16], Spark [21] and Myria [10]. In contrast to traditional database systems, where the computational complexity is dominated by the disk access time, the data now typically fits in main memory, and the dominant cost becomes that of communicating data and synchronizing among the servers in the cluster.

In this paper, we present a *worst-case analysis* of algorithms for processing of conjunctive queries (multiway join queries) on such massively parallel systems. Our analysis is based on the Massively Parallel Computation model, or MPC [4, 5]. MPC is a theoretical model where the computational complexity of an algorithm is characterized by both the *number of*

* This work is partially supported by NSF IIS-1247469, AitF 1535565, CCF-1217099 and CCF-1524246.

rounds (so the number of synchronization barriers) and the maximum amount of data, or *maximum load*, that each processor receives at every round.

The focus of our analysis on worst-case behavior of algorithms is a fundamentally different approach from previous work, where optimality of a parallel algorithm was defined for a specific input, or a specific family of inputs. Here we obtain upper bounds on the load of the algorithm across all possible types of input data. To give a concrete example, consider the simple join between two binary relations R and S of size M in bits (and m tuples), denoted $q(x, y, z) = R(x, z), S(y, z)$, and suppose that the number of servers is p. In the case where there is no data skew (which means in our case that the frequency of each value of the z variable in both R and S is at most m/p), it has been shown in [5] that the join can be computed in a single round with load $\tilde{O}(M/p)$ (where the notation $\tilde{O}$ hides a polylogarithmic factor depending on p), by simply hashing each tuple according to the value of the z variable. However, if the z variable is heavily skewed both in R and S (and in particular if there exists a single value of z), computing the query becomes equivalent to computing a cartesian product, for which we need $\Omega(M/p^{1/2})$ load. In this scenario, although for certain instances we can obtain better guarantees for the load, the heavily skewed instance is a *worst-case input*, in the sense that the lower bound $\Omega(M/p^{1/2})$ specifies the worst possible load that we may encounter. Our goal is to design algorithms for single or multiple rounds that are optimal with respect to such worst-case inputs and never incur larger load for any input.

Related Work. Algorithms for joins in the MPC model were previously analyzed in [4, 5]. In [4], the authors presented algorithms for one and multiple rounds on input data without skew (in particular when each value appears exactly once in each relation, which is a called a *matching database*). In [5], the authors showed that the HyperCube (HC) algorithm, first presented by Afrati and Ullman [2], can optimally compute any conjunctive query for a single round on data without skew. The work in [5] also presents one-round algorithms and lower bounds for skewed data but the upper and lower bounds do not necessarily coincide.

Several other computation models have been proposed in order to understand the power of MapReduce and related massively parallel programming paradigms [8, 13, 15, 1]. All these models identify the number of communication steps/rounds as a main complexity parameter, but differ in their treatment of the communication. Previous work [20, 14] has also focused on computing various graph problems in message-passing parallel models. In contrast to this work, where we focus on algorithms that require a constant number of rounds, the authors consider algorithms that need a large number of rounds.

Our setting and worst-case analysis can be viewed as the analogous version of the work of Ngo et al. [17] on worst-case optimal algorithms for multiway join processing. As we will show later, the worst-case instances for a given query q are different for the two settings in the case of one round, but coincide for all the families of queries we examine when we consider multiple rounds.

Our Contributions. We first present in Section 3 tight upper and lower bounds for the worst-case load of one-round algorithms for any full conjunctive query q without self-joins.[1] The optimal algorithm uses a different parametrization (share allocation) of the HyperCube algorithm for different parts of the input data, according to the values that are skewed. In

[1] The restriction to queries without self-joins is not limiting, since we can extend our result to queries with self-joins (by losing a constant factor) by treating copies of a relation as distinct relations. The parallel complexity for queries with projections is however an open question.

the case where all relation sizes are equal to M, the algorithm achieves an optimal load $\tilde{O}(M/p^{1/\psi^*(q)})$, where $\psi^*(q)$ is the *edge quasi-packing number* of the query q. An edge quasi-packing is an edge packing on any vertex-induced projection of the query hypergraph (in which we shrink hyperedges when we remove vertices).

In Section 4, we show that for any full conjunctive query q, any algorithm with a constant number of rounds requires a load of $\Omega(M/p^{1/\rho^*})$, where ρ^* is the *edge cover number*. We then present optimal (within a polylogarithmic factor) multi-round algorithms for several classes of join queries. Our analysis shows that some queries (such as the star query T_k) can be optimally computed using the optimal single-round algorithm from Section 3. However, other classes of queries, such as the cycle query C_k for $k \neq 4$, the line query L_k, or the Loomis-Whitney join LW_k require 2 or more rounds to achieve the optimal load. For example, we present an algorithm for the full query (or clique) K_k that uses $k-1$ rounds to achieve the optimal load (although it is open whether only 2 rounds are sufficient).

Finally, in Section 5 we present a surprising application of our results in the setting of external memory algorithms. In this setting, the input data does not fit into main memory, and the dominant cost of an algorithm is the I/O complexity: reading the data from the disk into the memory and writing data on the disk. In particular, we show that we can simulate an MPC algorithm in the external memory setting, and obtain almost-optimal (within a polylogarithmic factor) external memory algorithms for computing triangle queries; the same technique can be easily applied to other classes of queries.

2 Background

In this section, we introduce the MPC model and present the necessary terminology and technical tools that we will use later in the paper.

2.1 The MPC Model

We first review the *Massively Parallel Computation model (MPC)*, which allows us to analyze the performance of algorithms in parallel environments. In the MPC model, computation is performed by p servers, or processors, connected by a complete network of private channels. The computation proceeds in *steps*, or *rounds*, where each round consists of two distinct phases. In the *communication phase*, the servers exchange data, each by communicating with all other servers. In the *computation phase*, each server performs only local computation.

The input data of size M bits is initially uniformly partitioned among the p servers, that is, each server stores M/p bits of data. At the end of the execution, the output must be present in the union of the output of the p processors.

The execution of a parallel algorithm in the MPC model is captured by two parameters. The first parameter is the *number of rounds* r that the algorithm requires. The second parameter is the *maximum load* L, which measures the maximum amount of data (in bits) received by any server during any round.

All the input data will be distributed during some round, since we need to perform some computation on it. Thus, at least one server will receive at least data of size M/p. On the other hand, the maximum load will never exceed M, since any problem can be trivially solved in one round by simply sending the entire data to server 1, which can then compute the answer locally. Our typical loads will be of the form $M/p^{1-\varepsilon}$, for some parameter ε ($0 \leq \varepsilon < 1$) that depends on the query. For a similar reason, we do not allow the number of rounds to reach $r = p$, because any problem can be trivially solved in p rounds by sending

M/p bits of data at each round to server 1, until this server accumulates the entire data. In this paper we only consider the case $r = O(1)$.

2.2 Conjunctive Queries

In this paper we focus on a particular class of problems for the MPC model, namely computing answers to conjunctive queries over a database. We fix an input vocabulary $S_1, \ldots, S_\ell$, where each relation S_j has a fixed arity a_j; we denote $a = \sum_{j=1}^{\ell} a_j$. The input data consists of one relation instance for each symbol.

We consider full conjunctive queries (CQs) without self-joins, denoted as follows:

$$q(x_1, \ldots, x_k) = S_1(\ldots), \ldots, S_\ell(\ldots)\,.$$

The query is *full*, meaning that every variable in the body appears in the head (for example $q(x) = S(x, y)$ is not full), and *without self-joins*, meaning that each relation name S_j appears only once (for example $q(x, y, z) = S(x, y), S(y, z)$ has a self-join). We use $\mathsf{vars}(S_j)$ to denote the set of variables in the atom S_j, and $\mathsf{vars}(q)$ to denote the set of variables in all atoms of q. Further, k and ℓ denote the number of variables and atoms in q respectively. The *hypergraph* of a conjunctive query q is defined by introducing one node for each variable in the body and one hyperedge for each set of variables that occur in a single atom.

The *fractional edge packing* associates a non-negative weight u_j to each atom S_j such that for every variable x_i, the sum of the weights for the atoms that contain x_i does not exceed 1. We let $\mathsf{pk}(q)$ denote the set of all fractional edge packings for q. The *fractional covering number* τ^* is the maximum sum of weights over all possible edge packings, $\tau^*(q) = \max_{\mathbf{u} \in \mathsf{pk}(q)} \sum_j u_j$.

The *fractional edge cover* associates a non-negative weight w_j to each atom S_j, such that for every variable x_i, the sum of the weights of the atoms that contain x_i is at least 1. The *fractional edge cover number* ρ^* is the minimum sum of weights over all possible fractional edge covers. The notion of the fractional edge cover has been used in the literature [3, 17] to provide lower bounds on the worst-case output size of a query (and consequently the running time of join processing algorithms).

For any $\mathbf{x} \subseteq \mathsf{vars}(q)$, we define the *residual query* $q_{\mathbf{x}}$ as the query obtained from q by removing all variables $\mathbf{x}$, and decreasing the arity of each relation accordingly (if the arity becomes zero we simply remove the relation). For example, for the *triangle query* $q(x, y, z) = R(x, y), S(y, z), T(z, x)$, the residual query $q_{\{x\}}$ is $q_{\{x\}}(y, z) = R(y), S(y, z), T(z)$. Similarly, $q_{\{x,y\}}(z) = S(z), T(z)$. Observe that every fractional edge packing of q is also a fractional edge packing of any residual query $q_{\mathbf{x}}$, but the converse is not true in general.

We now define the *fractional edge quasi-packing* to be any edge packing of a residual query $q_{\mathbf{x}}$ of q, where the atoms that have only variables in $\mathbf{x}$ get a weight of 0. Denote by $\mathsf{pk}^+(q)$ the set of all edge quasi-packings. It is straightforward to see that $\mathsf{pk}(q) \subseteq \mathsf{pk}^+(q)$; in other words, any packing is a quasi-packing as well. The converse is not true, since for example $(1, 1, 0)$ is a quasi-packing for the triangle query, but not a packing. The *edge quasi-packing number* ψ^* is the maximum sum of weights over all edge quasi-packings:

$$\psi^*(q) = \max_{\mathbf{u} \in \mathsf{pk}^+(q)} \sum_j u_j = \max_{\mathbf{x} \subseteq \mathsf{vars}(q)} \max_{\mathbf{u} \in \mathsf{pk}(q_{\mathbf{x}})} \sum_j u_j\,.$$

2.3 Previous Results

Suppose that we are given a full CQ q, and input such that relation S_j has size M_j in bits (we use m_j for the number of tuples). Let $\mathbf{M} = (M_1, \ldots, M_\ell)$ be the vector of the relation

sizes. For a given fractional edge packing $\mathbf{u} \in \mathsf{pk}(q)$, we define as in [5]:

$$L(\mathbf{u}, \mathbf{M}, p) = \left(\frac{\prod_{j=1}^{\ell} M_j^{u_j}}{p} \right)^{1/\sum_{j=1}^{\ell} u_j} \tag{1}$$

Let us also define $L^{(q)}(\mathbf{M}, p) = \max_{\mathbf{u} \in \mathsf{pk}(q)} L(\mathbf{u}, \mathbf{M}, p)$. In our previous work [5], we showed that any algorithm that computes q in a single round with p servers must have load $L \geq L^{(q)}(\mathbf{M}, p)$. The instances used to prove this lower bound is the class of *matching databases*, which are instances where each value appears exactly once in the attribute of each relation. Hence, the above lower bound is not necessarily tight; indeed, as we will see in the next section, careful choice of skewed input instances can lead to a stronger lower bound.

The HyperCube algorithm. To compute conjunctive queries in the MPC model, we use the basic primitive of the *HyperCube (HC) algorithm*. The algorithm was first introduced by Afrati and Ullman [2], and was later called the *shares* algorithm; we use the name HC to refer to the algorithm with a particular choice of shares. The HC algorithm initially assigns to each variable x_i a *share* p_i, such that $\prod_{i=1}^{k} p_i = p$. Each server is then represented by a distinct point $\mathbf{y} \in \mathcal{P}$, where $\mathcal{P} = [p_1] \times \cdots \times [p_k]$; in other words, servers are mapped into a k-dimensional hypercube. The HC algorithm then uses k independently chosen hash functions $h_i : \{1, \ldots, n\} \to \{1, \ldots, p_i\}$ (where n is the domain size) and sends each tuple t of relation S_j to all servers in the destination subcube of t:

$$\mathcal{D}(t) = \{\mathbf{y} \in \mathcal{P} \mid \forall x_i \in \mathsf{vars}(S_j) : h_i(t[x_i]) = \mathbf{y}_i\}$$

where $t[x_i]$ denotes the value of tuple t at the position of the variable x_i. After the tuples are received, each server locally computes q for the subset of the input that it has received.

If the input data has no skew, the above vanilla version of the HC algorithm is optimal for a single round. The lemma below presents the specific conditions that define skew, and will be frequently used throughout the paper.

▶ **Lemma 1** (Load Analysis for HC [5]). *Let $\mathbf{p} = (p_1, \ldots, p_k)$ be the optimal shares of the HC algorithm. Suppose that for every relation S_j and every tuple t over the attributes $U \subseteq [a_j]$ we have that the frequency of t in relation S_j is $m_{S_j}(t) \leq m_j / \prod_{i \in U} p_i$. Then with high probability the maximum load per server is $\tilde{O}(L^{(q)}(\mathbf{M}, p))$.*

3 One-Round Algorithms

In this section, we present tight upper and lower bounds for the worst-case load of one-round algorithms that compute conjunctive queries. Thus, we identify the database instances for which the behavior in a parallel setting is the worst possible. Surprisingly, these instances are often different from the ones that provide a worst-case running time in a non-parallel setting.

As an example, consider the triangle query $C_3 = R(x, y), S(y, z), T(z, x)$, where all relations have m tuples (and M in bits). It is known from [3] that the class of inputs that will give a worst-case output size, and hence a worst-case running time, is one where each relation is a $\sqrt{m} \times \sqrt{m}$ fully bipartite graph. In this case, the output has $m^{3/2}$ tuples. The load needed to compute C_3 on this input in a single round is $\Omega(M/p^{2/3})$, and can be achieved by using the HyperCube algorithm [4] with shares $p^{1/3}$ for each variable. Now, consider the instance where relations R, T have a single value at variable x, which participates in all the m tuples in R and T; S is a matching relation with m tuples. In this case, the output has m

tuples (and so M bits), and thus is smaller than the worst-case output. However, as we will see next, we can show that any one-round algorithm that computes the triangle query for the above input structure requires $\Omega(M/p^{1/2})$ maximum load.

3.1 An Optimal Algorithm

We present here a worst-case optimal one-step algorithm that computes a conjunctive query q. Recall that the HC algorithm achieves an optimal load on data without skew [5]. In the presence of skew, we will distinguish different cases, and for each case we will apply a different parametrization of the HC algorithm, using different shares.

We say that a value h in relation S_j is a *heavy hitter* in S_j if the frequency of this particular value in S_j, denoted $m_{S_j}(h)$, is at least m_j/p, where m_j is the number of tuples in the relation. Given an output tuple t, we say that t is *heavy at variable* x_i if the value $t[x_i]$ is a heavy hitter in at least one of the atoms that include variable x_i.

We can now classify each tuple t in the output depending on the positions where t is heavy. In particular, for any $\mathbf{x} \subseteq \mathsf{vars}(q)$, let $q^{[\mathbf{x}]}(I)$ denote the subset of the output that includes only the output tuples that are heavy at exactly the variables in $\mathbf{x}$. Observe that the case $q^{[\emptyset]}(I)$ denotes the case where the tuples are light at all variables; we know from an application of Lemma 1 that this case can be handled by the standard HC algorithm. For each of the remaining $2^k - 1$ possible sets $\mathbf{x} \subseteq \mathsf{vars}(q)$, we will run a different variation of the HC algorithm with different shares, which will allow us to compute $q^{[\mathbf{x}]}(I)$ with the appropriate load. Our algorithm will compute all the partial answers in parallel for each $\mathbf{x} \subseteq \mathsf{vars}(q)$, and thus requires only a single round.

The key idea is to apply the HC algorithm by giving a non-trivial share only to the variables that are not in $\mathbf{x}$; in other words, every variable in $\mathbf{x}$ gets a share of 1. In particular, we will assign to the remaining variables the shares we would assign if we would execute the HC algorithm for the residual query $q_{\mathbf{x}}$. We will thus choose the shares by assigning $p_i = p^{e_i}$ for each $x_i \in \mathbf{x}$ and solving the following linear program:

$$\begin{aligned} \text{minimize} \quad & \lambda \\ \text{subject to} \quad & \sum_{i:x_i \notin \mathbf{x}} -e_i \geq -1 \\ \forall j \text{ s.t. } S_j \in \mathsf{atoms}(q_{\mathbf{x}}) : & \sum_{i:x_i \in \mathsf{vars}(S_j) \setminus \mathbf{x}} e_i + \lambda \geq \mu_j \\ \forall i \text{ s.t. } x_i \notin \mathbf{x} : & e_i \geq 0, \quad \lambda \geq 0 \end{aligned} \tag{2}$$

For each variable $x_i \in \mathbf{x}$, we set $e_i = 0$ and thus the share is $p_i = 1$. We next present the analysis of the load for the above algorithm.

▶ **Theorem 2.** *Any full conjunctive query q with input relation sizes $\mathbf{M}$ can be computed in the MPC model in a single round using p servers with maximum load*

$$L = \tilde{O}\left(\max_{\mathbf{x} \subseteq \mathit{vars}(q)} L^{(q_{\mathbf{x}})}(\mathbf{M}, p)\right).$$

Proof. Let us fix a set of variables $\mathbf{x} \subseteq \mathsf{vars}(q)$; we will show that the load of the algorithm that computes $q^{[\mathbf{x}]}(I)$ is $\tilde{O}(L^{(q_{\mathbf{x}})}(\mathbf{M}, p))$. The upper bound then follows from the fact that we are running in parallel algorithms for all partial answers.

Indeed, let us consider how each relation S_j is distributed using the shares assigned. We distinguish two cases. If an atom S_j contains variables that are only in $\mathbf{x}$, then the whole

relation will be broadcast to all the p servers. However, observe that the part of S_j that contributes to $q^{[\mathbf{x}]}(I)$ is of size at most p^{a_j}, where a_j is the arity of the relation.

Otherwise, we will show that for every tuple J of values over variables $\mathbf{v} \subseteq \mathsf{vars}(S_j)$, we have that the frequency of J is at most $m_j / \prod_{i:x_i \in \mathbf{v}} p_i$. Indeed, if $\mathbf{v}$ contains only variables from $\mathbf{x}$, then by construction $\prod_{i:x_i \in \mathbf{v}} p_i = 1$; we observe then that the frequency is always at most m_j. If $\mathbf{v}$ contains some variable $x_i \in \mathbf{v} \setminus \mathbf{x}$, then the tuple J contains at position x_i a value that appears at most m_j/p times in relation S_j, and since $\prod_{i:x_i \in \mathbf{v}} p_i \leq p$ the claim holds. We can now apply Lemma 1 to obtain that for relation S_j, the load will be $\tilde{O}(M_j/(\prod_{i:x_i \in \mathsf{vars}(S_j)\setminus \mathbf{x}} p_i))$. Summing over all atoms in the residual query $q_{\mathbf{x}}$, and assuming that $m_j \gg p$ (and in particular that p^{a_j} is always much smaller than the load), we obtain that the load will be $\tilde{O}(\max_{j:S_j \in \mathsf{atoms}(q_{\mathbf{x}})} M_j/(\prod_{i:x_i \in \mathsf{vars}(S_j)\setminus \mathbf{x}} p_i))$, which by an LP duality argument is equal to $\tilde{O}(L^{(q_{\mathbf{x}})}(\mathbf{M}, p))$. ◀

When all relation sizes are equal, that is, $M_1 = M_2 = \cdots = M_\ell = M$, the formula for the maximum load becomes $\tilde{O}(M/p^{1/\psi^*(q)})$, where $\psi^*(q)$ is the edge quasi-packing number, which we have defined as $\psi^*(q) = \max_{\mathbf{x} \subseteq \mathsf{vars}(q)} \max_{\mathbf{u} \in \mathsf{pk}(q_{\mathbf{x}})} \sum_j u_j$. We will discuss about the quantity $\psi^*(q)$ in detail in Section 3.3. We will see next how the above algorithm applies to the triangle query C_3.

▶ **Example 3.** We will describe first how the algorithm works when each relation has size M (and m tuples). There are three different share allocations, for each choice of heavy variables (all other cases are symmetrical).

$\mathbf{x} = \emptyset$: we consider only tuples with values of frequency $\leq m/p$. The HC algorithm will assign a share of $p^{1/3}$ to each variable, and the maximum load will be $\tilde{O}(M/p^{2/3})$.

$\mathbf{x} = \{x\}$: the tuples have a heavy hitter value at variable x, either in relation R or T or in both. The algorithm will give a share of 1 to x, and shares of $p^{1/2}$ to y and z. The load will be $\tilde{O}(M/p^{1/2})$.

$\mathbf{x} = \{x, y\}$: both x and y are heavy. In this case we broadcast the relation $R(x, y)$, which will have size at most p^2, and assign a share of p to z. The load will be $\tilde{O}(M/p)$.

Notice finally that the case where $\mathbf{x} = \{x, y, z\}$ can be handled by broadcasting all necessary information. The load of the algorithm is the maximum of the above quantities, thus $\tilde{O}(M/p^{1/2})$. When the size vector is $\mathbf{M} = (M_1, M_2, M_3)$, the load achieved becomes $\tilde{O}(L)$, where: $L = \max\left\{\frac{M_1}{p}, \frac{M_2}{p}, \frac{M_3}{p}, \sqrt{\frac{M_1 M_2}{p}}, \sqrt{\frac{M_2 M_3}{p}}, \sqrt{\frac{M_1 M_3}{p}}\right\}$.

3.2 Lower Bounds

We present here a worst-case lower bound for the load of one-step algorithms for computing conjunctive queries in the MPC model, when the information known is the cardinality statistics $\mathbf{M} = (M_1, \ldots, M_\ell)$. The lower bound matches the upper bound in the previous section, hence proving that the one-round algorithm is worst-case optimal. We give a self-contained proof of the result in the full version of this paper, but many of the techniques used can be found in previous work [4, 5], where we proved lower bounds for skew-free data and for input data with known information about the heavy hitters.

▶ **Theorem 4.** *Fix cardinality statistics* $\mathbf{M}$ *for a full conjunctive query* q*. Consider any deterministic MPC algorithm that runs in one communication round on* p *servers and has maximum load* L *in bits. Then, for any* $\mathbf{x} \subseteq \mathsf{vars}(q)$*, there exists a family of (random) instances for which the load* L *will be:*

$$L \geq \min_j \frac{1}{4a_j} \cdot L^{(q_{\mathbf{x}})}(\mathbf{M}, p)\,.$$

Since $a_j \geq 1$, Theorem 4 implies that for any query q there exists a family of instances such that any one-round algorithm that computes q must have load $\Omega(\max_{\mathbf{x} \subseteq \mathsf{vars}(q)} L^{(q_{\mathbf{x}})}(\mathbf{M}, p))$.

3.3 Discussion

We present here several examples for the load of the one-round algorithm for various classes of queries, and also discuss the edge quasi-packing number $\psi^*(q)$ and its connection with other query-related quantities.

Recall that we showed that when all relation sizes are equal to M, the load achieved is of the form $\tilde{O}(M/p^{1/\psi^*(q)})$, where $\psi^*(q)$ is the quantity that maximizes the sum of the weights of the edge quasi-packing. $\psi^*(q)$ is in general different from both the fractional covering number $\tau^*(q)$, and from the fractional edge cover number $\rho^*(q)$. Indeed, for the triangle query C_3 we have that $\rho^*(C_3) = \tau^*(C_3) = 3/2$, while $\psi^*(C_3) = 2$. Here we should remind the reader that τ^* describes the load for one-round algorithms on data without skew, which is $O(M/p^{1/\tau^*(q)})$. Also, ρ^* characterizes the maximum possible output of a query q, which is $M^{\rho^*(q)}$. We can show the following relation between the three quantities:

▶ **Lemma 5.** *For every conjunctive query q, $\psi^*(q) \geq \max\{\tau^*(q), \rho^*(q)\}$.*

Proof. Since any edge packing is also an edge quasi-packing, it is straightforward to see that $\tau^*(q) \leq \psi^*(q)$ for every query q.

To show that $\rho^*(q) \leq \psi^*(q)$, consider the optimal (minimum) edge cover $\mathbf{u}$; we will show that this is also an edge quasi-packing. First, observe that for every atom S_j, there must exist at least one variable $x \in \mathsf{vars}(S_j)$ such that $\sum_{j:x \in \mathsf{vars}(S_j)} u_j = 1$. Indeed, suppose that for every variable in S_j we have that the sum of the weights strictly exceeds 1; then, we can obtain a better edge cover by slightly decreasing u_j, which is a contradiction.

Now, let $\mathbf{x}$ be the set of variables such that their cover in $\mathbf{u}$ strictly exceeds 1, and consider the residual query $q_{\mathbf{x}}$. By our previous claim, every relation in q is still present in $q_{\mathbf{x}}$, since every relation includes a variable with cover exactly one. Further, for every variable $x \in \mathsf{vars}(q_{\mathbf{x}})$ we have $\sum_{j:x \in \mathsf{vars}(S_j)} u_j = 1$, and hence $\mathbf{u} \in \mathsf{pk}(q_{\mathbf{x}})$. ◀

In Table 1 we have computed the quantities τ^*, ρ^*, ψ^* for several classes of queries of interest: the star query T_k, the spiked star query SP_k, the cycle query C_k, the line query L_k, the Loomis-Whitney join LW_k, the generalized semi-join query W_k and the clique (or full) query K_k. We next present some of these queries in more detail.

▶ **Example 6.** Consider the star query T_k, which generalizes the simple join between relations. As we can see, the optimal edge packing cannot be more than 1, since every relation includes the variable z. To obtain the maximum edge quasi-packing, we simply consider the residual query q_z that removes the common variable z: then, we can pack each relation with weight one, thus achieving a sum of k. Notice that this is an example which shows that τ^* and ψ^* cannot be within a constant factor.

▶ **Example 7.** Consider the full/clique query K_k, which includes all possible binary relations among the k variables. Here the optimal edge packing is achieved by assigning a weight of $1/(k-1)$ to each relation; the corresponding share allocation for the HC algorithm assigns an equal share of $p^{1/k}$ to each variable. For the optimal edge quasi-packing, consider the residual query $(K_k)_{x_1}$, and notice that it includes $(k-1)$ unary relations, one for each of $x_2, \ldots, x_k$. Hence, we can obtain an edge packing by assigning a weight of 1 to each, which shows that $\psi^*(K_k) = k$.

Table 1 Computing the optimal edge packing τ^*, edge cover ρ^* and edge quasi-packing ψ^* for several classes of conjunctive queries.

conjunctive query	τ^*	ρ^*	ψ^*
$T_k = \bigwedge_{j=1}^{k} S_j(z, x_j)$	1	k	k
$SP_k = \bigwedge_{i=1}^{k} R_i(z, x_i), S_i(x_i, y_i)$	k	$k+1$	$k+1$
$C_k = \bigwedge_{j=1}^{k} S_j(x_j, x_{(j \bmod k)+1})$	$k/2$	$k/2$	$\lceil 2(k-1)/3 \rceil$
$L_k = \bigwedge_{j=1}^{k} S_j(x_{j-1}, x_j)$	$\lceil k/2 \rceil$	$\lceil (k+1)/2 \rceil$	$\lceil 2k/3 \rceil$
$LW_k = \bigwedge_{I \subseteq [k], \lvert I \rvert = k-1} S_I(\bar{x}_I)$	$k/(k-1)$	$k/(k-1)$	2
$W_k = R(x_1, \ldots, x_k) \bigwedge_{j=1}^{k} S_j(x_j)$	k	1	k
$K_k = \bigwedge_{1 \le i < j \le k} S_{i,j}(x_i, x_j)$	$k/2$	$k/2$	k

▶ **Example 8.** Consider the cycle query C_k. The optimal edge packing assigns a weight of $1/2$ to each edge; the corresponding share allocation for the HC algorithm gives an equal share of $p^{1/k}$ to each variable.

To find the best $\mathbf{x}$ for the optimal edge quasi-packing, we will pick every third variable: $x_1, x_4, \ldots$. This creates $\lfloor k/3 \rfloor$ copies of the query $S_1(x_1), S_2(x_1, x_2), S_3(x_2)$, which has an edge packing of size 2 (assign weight 1 to S_1, S_3). If $k = 3m$ or $k = 3m+1$, these copies cover the whole query. If $k = 3m + 2$, we can add one more edge with weight 1 to the packing.

4 Multi-round Algorithms

In this section, we present algorithms for multi-round computation of several conjunctive queries in the case where the relation sizes are all equal to M. We also prove a lower bound that proves that they are (almost) optimal.

4.1 Multi-round Lower Bound

We prove here a general lower bound for any algorithm that computes conjunctive queries using a constant number of rounds. Observe that the lower bound is expressed in terms of number of tuples (and not bits); our upper bounds will be expressed in terms of bits, and thus will be a $\log(n)$ factor away from the lower bound, where n is the domain size.

▶ **Theorem 9.** *Let q be a conjunctive query. Then, there exists a family of instances where relations have the same size M in bits (and m in tuples) such that every algorithm that computes q with p servers using a constant number of rounds requires load $\Omega(m/p^{1/\rho^*(q)})$.*

Proof. In order to prove the lower bound, we will use a family of instances that give the maximum possible output when every input relation has at most m tuples, which is $m^{\rho^*(q)}$ (see [3]). We also know how we can construct such a worst-case instance: for each variable x_i we assign an integer n_i (which corresponds to the domain size of the variable), and we define each relation as the cartesian product of the domains of the variables it includes: $\times_{i:x_i \in \mathsf{vars}(S_j)}[n_i]$. The output size then will be $\prod_i n_i = m^{\rho^*(q)}$ (using a LP duality argument).

We now define the following random instance I as input for the query q: for each relation S_j, we choose each tuple from the full cartesian product of the domains independently at random with probability $1/2$. It is straightforward to see that the expected size of the output is $E[|q(I)|] = (1/2)^{\beta} \prod_i n_i$, where β is the maximum number of relations where any variable

occurs (and thus a constant depending on the query). Using Chernoff's bound we can claim an even stronger result: the output size will be $\Theta(m^{\rho^*(q)})$ with high probability (the failure probability is exponentially small in m).

Now, assume that algorithm $\mathcal{A}$ computes q with load L (in bits) in r rounds. Then, each server receives at most $L' = r \cdot L$ bits. Fix some server and let msg be the whole sequence of bits received by this server during the computation; hence, $|\mathsf{msg}| \leq L'$. We will next compute how many tuples from S_j are known by the server, denoted $K_{\mathsf{msg}}(S_j)$. W.l.o.g. we can assume that all L' bits of msg contain information from relation S_j.

We will show that the probability of the event $K_{\mathsf{msg}}(S_j) > (1+\delta)L'$ is exponentially small on δ. Let $m_j = \prod_{i:x_i \in \mathsf{vars}(S_j)} n_i \leq m$. Observe first that the total number of message configurations of size L' is at most $2^{L'}$. Also, since the size of the full cartesian product is m_j, msg can encode at most $2^{m_j-(1+\delta)L'}$ relations S_j (if $m_j < (1+\delta)L'$, then trivially the probability of the event is zero, and S_j will have "few" tuples). It follows that

$$P(K_{\mathsf{msg}} > (1+\delta)L') < 2^{L'} \cdot 2^{m_j-(1+\delta)L'} \cdot (1/2)^{m_j} = (1/2)^{\delta L'}.$$

So far we have shown that with high probability each server knows at most L' tuples from each relation S_j, and further that the total number of output tuples is $\Theta(m^{\rho^*(q)})$. However, if a server knows L' tuples from each relation, using the AGM bound from [3], it can output at most $(rL)^{\rho^*(q)}$ tuples. The result follows by summing over the output of all p servers, and using the fact that the algorithm has only a constant number of rounds. ◀

The theorem implies that whenever $\psi^*(q) = \rho^*(q)$ the one-round algorithm is essentially worst-case optimal, and using more rounds will not result in an algorithm with better load. As a result, and following our discussion in the previous section, the classes of queries T_k and SP_k can be optimally computed in a single round. This may seem counterintuitive, but recall that we study worst-case optimal algorithms; there may be instances where using more rounds is desirable, but our goal is to match the load for the worst such instance.

We will next present algorithms that match (within a logarithmic factor) the above lower bound using strictly more than one round. We start with the algorithm for the triangle query C_3, in order to demonstrate our novel technique and prove a key result (Lemma 10) that we will use later in the section.

4.2 Warmup: Computing Triangles in 2 Rounds

The main component of the algorithm that computes triangles is a parallel algorithm that computes the join $S_1(x,z), S_2(y,z)$ in a single round, for the case where skew appears exclusively in one of the two relations. If the relations have size M_1, M_2 respectively, then we have shown that the load can be as large as $\sqrt{M_1 M_2/p}$. However, in the case of one-sided skew, we can compute the join with maximum load only $\tilde{O}(\max\{M_1, M_2\}/p)$.

▶ **Lemma 10.** *Let $q = S_1(x,z), S_2(y,z)$, and let m_1 and m_2 be the relation sizes (in tuples) of S_1, S_2 respectively. Let $m = \max\{m_1, m_2\}$. If the degree of every value of the variable z in S_1, $m_{S_1}(z)$, is at most m/p, then we can compute q in a single round with p servers and load (in bits) $\tilde{O}(M/p)$, where $M = 2m\log(n)$ (n is the domain size).*

Proof. We say that a value h is a heavy hitter in S_2 if the degree of h in S_2 is $m_{S_2}(h) > m/p$. By our assumption, there are no heavy hitters in relation S_1.

For the values h that are not heavy hitters in S_2, we can compute the join by applying the standard HC algorithm (which is a hash-join that assigns a share of p to z); the load analysis of Lemma 1 will give us a load of $\tilde{O}(M/p)$ with high probability.

For every heavy hitter h, the algorithm computes the subquery $q[h/z] = S_1(x,h), S_2(y,h)$, which is equivalent to computing the residual query $q_z = S_1'(x), S_2'(y)$, where $S_1'(x) = S_1(x,h)$ and $S_2'(y) = S_2(y,h)$. We know that $|S_2'| = m_{S_2}(h)$ and $|S_1'| \leq m/p$ by our assumption. The algorithm now allocates $p_h = \lceil p \cdot m_{S_2}(h)/m \rceil$ exclusive servers to compute $q[h/z]$ for each heavy hitter h. To compute $q[h/z]$ with p_h servers, we simply use the simple broadcast join that assigns a share of p to variable x and 1 to y. A simple analysis will give us that the load (in tuples) for each heavy hitter h is

$$\tilde{O}\left(\frac{|S_2'|}{p_h} + |S_1'|\right) = \tilde{O}\left(\frac{m_{S_2}(h)}{p \cdot m_{S_2}(h)/m} + m/p)\right) = \tilde{O}(m/p)\,.$$

Finally, observe that the total number of servers we need is $\sum_h p_h \leq 2p$, hence we have used an appropriate amount of the available p servers. ◀

Thus, we can optimally compute joins in a single round in the presence of one-sided skew. We can apply Lemma 10 to obtain a useful corollary for the *semi-join* query $q = R(z), S(y,z)$. Indeed, notice that we can extend R to a binary relation $R'(x,z)$, where x is a dummy variable that takes a single value; then, the semi-join becomes essentially a join, where R' has no skew, since the degree of z in R' will be always one. Consequently:

▶ **Corollary 11.** *Consider the semi-join query $q = R(z), S(y,z)$, and let M_1 and M_2 be the relation sizes of R, S respectively in bits. Then we can compute q in a single round with p servers and load $\tilde{O}(\max\{M_1, M_2\}/p)$.*

We now outline the algorithm for computing triangles using two rounds. The central idea in the algorithm is to identify the values that create skew in the computation, and spread this computation into more rounds.

▶ **Theorem 12.** *The triangle query $C_3 = S_1(x_1,x_2), S_2(x_2,x_3), S_3(x_3,x_1)$ on input with sizes $M_1 = M_2 = M_3 = M$ can be computed by an MPC algorithm in 2 rounds with $\tilde{O}(M/p^{2/3})$ load, under any input data distribution.*

Proof. We say that a value h is heavy if for some relation S_j, we have $m_j(h) > m/p^{1/3}$. We first compute the answers for the tuples that are not heavy at any variable. Indeed, if for every value we have that the degree is at most $m/p^{1/3}$, then the load analysis (Lemma 1) tells us that we can compute the output in a single round with load $\tilde{O}(M/p^{2/3})$ using the HC algorithm that allocates a share of $p^{1/3}$ to each variable.

Thus, it remains to output the tuples for which at least one variable has a heavy value. Without loss of generality, consider the case where variable x_1 has heavy values and observe that there are at most $2p^{1/3}$ such heavy values for x_1 ($p^{1/3}$ for S_1 and $p^{1/3}$ for S_3). For each heavy value h, we assign an *exclusive* set of $p' = p^{2/3}$ servers to compute the query $q[h/x_1] = S_1(h,x_2), S_2(x_2,x_3), S_3(x_3,x_1)$, which is equivalent to computing the residual query $q' = S_1'(x_2), S_2(x_2,x_3), S_3'(x_3)$.

To compute q' with p' servers, we use 2 rounds. In the first round, we compute in parallel the semi-join queries $S_{12}(x_2,x_3) = S_1'(x_2), S_2(x_2,x_3)$ and $S_{23}(x_2,x_3) = S_2(x_2,x_3), S_3'(x_3)$. Since $|S_1'| \leq m$ and $|S_2'| \leq m$, we can apply Corollary 11 for semi-join computation to obtain that we can achieve this computation with load (in tuples) $\tilde{O}(m/p') = \tilde{O}(m/p^{2/3})$. Observe that the intermediate relations S_{12}, S_{23} have size at most m. In the second round, we simply perform the intersection of the relations S_{12}, S_{23}; this can be achieved with tuple load $O(m/p') = O(m/p^{2/3})$.[2] ◀

[2] Observe that the load for computing the intersection of two or more relations does not have any additional logarithmic factors.

Notice that the 2-round algorithm achieves a better load than the 1-round algorithm in the worst-case scenario. Indeed, in the previous section we proved that there exist instances for which we can not achieve load better than $O(M/p^{1/2})$ in a single round. By using an additional round, we can beat this bound and achieve a better load. This confirms our intuition that with more rounds we can reduce the maximum load. Moreover, observe that the load achieved matches the multi-round lower bound (within a polylogarithmic factor).

4.3 Computing General CQs

We now generalize the ideas of the above example, and extend our results to several standard classes of conjunctive queries. Throughout this section, we assume that all relations have the same size M in bits (and m in tuples). We present in detail optimal multiround algorithms for odd and even cycles, which both achieve a maximum load of $\tilde{O}(M/p^{2/k})$ for C_k. The algorithm uses as a component an optimal algorithm that computes the line query L_k.

▶ **Lemma 13.** *The line query $L_k = S_1(x_0, x_1), S_2(x_1, x_2), \ldots, S_k(x_{k-1}, x_k)$ can be computed by an MPC algorithm with a constant number of rounds and load $\tilde{O}(M/p^{1/\lceil (k+1)/2 \rceil})$.*

We then briefly present our algorithmic results for Loomis-Whitney joins and Clique queries; the detailed proofs of the desired load are in the full version of this paper.

4.3.1 Odd Cycles

We will first show how we can compute any odd cycle C_k; the algorithm is a generalization of the method for computing triangle queries presented as a warmup example.

We say that a value h is *heavy* for variable x_i if for relation S_{i-1} or S_i, we have $m_i(h) > m/p^{1/k}$ or $m_{i-1}(h) > m/p^{1/k}$. We first compute the answers for the tuples that are not heavy at any position. Lemma 1 implies that we can compute the output in a single round with load $\tilde{O}(M/p^{2/k})$, by applying the vanilla HC algorithm for cycles, where each variable has equal share $p^{1/k}$.

We next compute the tuples that are heavy at variable x_1 (we similarly do this for every variable x_i); observe that there are at most $2p^{1/k}$ such values. For each such heavy value h, we will assign an exclusive number of $p' = p^{1-1/k}$ servers, such that the total number of servers we use is $(2p^{1/k}) \cdot p' = \Theta(p)$, and using these servers we will compute the query $q[h/x_1] = S_1(h, x_2), \ldots, S_k(x_k, h)$, which amounts to computing the residual query $q' = q_{x_1}$:

$$q' = S_1'(x_2), S_2(x_2, x_3), \ldots, S_{k-1}(x_{k-1}, x_k), S_k'(x_k)\,.$$

To compute q' with p' servers we need two rounds of computation. In the first round, we compute in parallel the two semi-joins

$$S_{1,2}(x_2, x_3) = S_1'(x_2), S_2(x_2, x_3), \qquad S_{k,k-1}(x_{k-1}, x_k) = S_{k-1}(x_{k-1}, x_k), S_k'(x_k)$$

which can be achieved with tuple load $\tilde{O}(m/p') = \tilde{O}(m/p^{1-1/k})$, since $|S_1'| \leq m$ and $|S_k'| \leq m$ (by applying Corollary 11). Since for any $k \geq 3$ we have $1 - 1/k \geq 2/k$, the load for the first round will be $\tilde{O}(M/p^{2/k})$. For the second round, we compute the query

$$q'' = S_{1,2}(x_2, x_3), S_3(x_3, x_4), \ldots, S_{k-1}(x_{k-1}, x_k), S_{k,k-1}(x_{k-1}, x_k)$$

which is equivalent to computing the line query L_{k-2}, where each relation has size at most m; we know from Lemma 13 that we can compute such a query with tuple load $\tilde{O}(m/p'^{1/\lceil (k-1)/2 \rceil})$ using multiple rounds. For the final step of the proof, recall that $p' = p^{1-1/k}$. Then:

$$\frac{k-1}{k} \cdot \frac{1}{\lceil (k-1)/2 \rceil} = \frac{k-1}{k} \cdot \frac{2}{k-1} = 2/k\,.$$

Thus, the load for the second round will be $\tilde{O}(M/p^{2/k})$ as well.

4.3.2 Even Cycles

For even length cycles, our previous argument does not work, and we have to use a different approach. We say that a value h is δ-heavy, for some $\delta \in [0,1]$, if the degree of h is at least m/p^δ in some relation. We distinguish two different cases:

1. Suppose that there exist two variables $x_i, x_{i'}$ such that $(i - i')$ is an odd number, x_i is δ-heavy, $x_{i'}$ is δ'-heavy, and $\delta + \delta' \leq 2/k$. Observe that there are at most $p^{\delta+\delta'} \leq p^{2/k}$ such pairs of heavy values: for each such pair, we assign $p' = p^{1-2/k}$ explicit servers to compute the residual query $q' = (C_k)_{x_i,x_j}$ in two rounds. We now consider two subcases. If $i' = i+1$, then $x_i, x_{i'}$ belong in the same relation S_i. Then, by performing the semi-join computations in the first round, we reduce the computation of the next rounds to the residual query L_{k-3}, which requires tuple load $\tilde{O}(m/p'^{1/\lceil (k-2)/2 \rceil}) = \tilde{O}(m/p^{2/k})$, since k is even. Otherwise, if $x_i, x_{i'}$ are not in the same relation, we still do the semi-joins in the first round, and then notice that in the subsequent rounds we need to compute the cartesian product of two line queries, L_α, L_β, where $\alpha + \beta = k - 4$ and both are odd numbers. To perform this cartesian product, we will split the p' servers into a $p^{(\alpha+1)/k} \times p^{(\beta+1)/k}$ grid, and within each row/column compute the line queries. Then, the tuple load will be $\tilde{O}(m/p^{((\alpha+1)/k)\cdot(1/\lceil(\alpha+1)/2\rceil)}) = \tilde{O}(m/p^{((\beta+1)/k)\cdot(1/\lceil(\beta+1)/2\rceil)}) = \tilde{O}(m/p^{2/k})$.
2. Otherwise, define δ_{even} as the largest number in $[0,1]$ such that for every even variable the frequency is at most $m/p^{\delta_{even}}$. Similarly define δ_{odd}. Since we do not fall in the previous case, it must be that $\delta_{even} + \delta_{odd} \geq 2/k$. W.l.o.g. assume that $\delta_{even} \geq \delta_{odd}$. Then, consider the HC algorithm with the following share allocation: for odd variables assign $p_o = p^{\delta_{odd}}$, and for even variables assign $p_e = p^{2/k - \delta_{odd}}$. Since the odd variables have degree at most $m/p^{\delta_{odd}}$, there are no skewed values there. As for the even variables, their degree is at most $m/p^{\delta_{even}} \leq m/p^{2/k-\delta_{odd}} = m/p_e$. Hence, the tuple load achieved will be $\tilde{O}(m/(p_o p_e) = \tilde{O}(m/p^{2/k})$. In the case where p_e is ill-defined because $\delta_{odd} > 2/k$, we also have that $\delta_{even} > 2/k$ and in this case we can just apply the standard HC algorithm that assigns a share of $p^{1/k}$ to every variable.

4.3.3 Other Conjunctive Queries

For the Loomis-Whitney (LW) join, the algorithmic idea is the same as the one we used for even cycles (notice that LW_3 is the triangle query C_3).

▶ **Lemma 14.** *The LW join $LW_k = S_1(x_2, \ldots, x_k), S_2(x_1, x_3, \ldots, x_k), \ldots, S_k(x_1, \ldots, x_{k-1})$ can be computed by an MPC algorithm in 2 rounds with load $\tilde{O}(M/p^{1-1/k})$.*

For the clique queries, we have the following result:

▶ **Lemma 15.** *The clique query $K_k = \bigwedge_{1 \leq i < j \leq k} S_{i,j}(x_i, x_j)$ can be computed by an MPC algorithm in $k-1$ rounds with load $\tilde{O}(M/p^{2/k})$ for any $k \geq 3$.*

Finally, we show an almost optimal algorithm for queries q that contain an atom which includes all the variables in the body of q.

▶ **Lemma 16.** *Let q be a query that contains an atom R, such that $\mathsf{vars}(R) = \mathsf{vars}(q)$. Then, q can be computed by an MPC algorithm with $\tilde{O}(M/p)$ load.*

Notice that the generalized semi-join query W_k satisfies the property of the above lemma, and hence we can compute W_k with load $\tilde{O}(M/p)$ using two rounds (while using one round the load is $\Omega(M/p^{1/k})$).

5 Applications to the External Memory Model

In the external memory model, we model computation in the setting where the input data does not fit into main memory, and the dominant cost is reading the data from the disk into the memory and writing data on the disk.

Formally, we have an external memory (disk) of unbounded size, and an internal memory (main memory) that consists of W *words*.[3] The processor can only use data stored in the internal memory to perform computation, and data can be moved between the two memories in blocks of B consecutive words. The *I/O complexity* of an algorithm is the number of input/output blocks that are moved during the algorithm, both from the internal memory to the external one, and vice versa.

The external memory model has been recently used in the context of databases to analyze algorithms for large datasets that do not fit in the main memory, with the main application being *triangle listing* [6, 12, 18, 11]. In this setting, the input is an undirected graph, and the goal is to list all triangles in the graph. In [18] and [11], the authors consider the related problem of *triangle enumeration*, where instead of listing triangles (and hence writing them to the external memory), for each triangle in the output we call an *emit*() function. The best result comes from [11], where the authors design a deterministic algorithm that enumerates triangles in $O(|E|^{3/2}/(\sqrt{W}B))$ I/Os, where E is the number of edges in the graph. The authors in [11] actually consider a more general class of join problems, the so-called *Loomis-Whitney enumeration*. In [19], the author presents external memory algorithms for enumerating subgraph patterns in graphs other than triangles.

The problem we consider in the context of external memory algorithms is a generalization of triangle enumeration. Given a full conjunctive query q, we want to *enumerate* all possible tuples in the output, by calling the *emit*() function for each tuple in the output of query q. We assume that each tuple in the input can be represented by a single word.

5.1 Simulating an MPC Algorithm

We will show how a parallel algorithm in the tuple-based MPC model can help us construct an external memory algorithm. The *tuple-based MPC model* is a restriction of the MPC model, where only tuples from subqueries of q can be communicated, and moreover the communication can take a very specific form: each tuple t during round k is sent to a set of servers $\mathcal{D}(t, k)$, where $\mathcal{D}$ depends only on the data statistics that are initially available to the algorithm. Such statistical information is the size of the relations, or information about the heavy hitters in the data.[4] All of the algorithms that we have presented so far in the previous sections satisfy the above assumption.

The idea behind the construction is that the distribution of the data to the servers can be used to decide which input data will be loaded into memory; hence, the load L will correspond to the size of the internal memory W. Similarities between hash-join algorithms used for parallel processing and the variants of hash-join used for out-of-core processing have been already known, where the common theme is to create partitions and then process them one at a time. Here we generalize this idea to the processing of any conjunctive query in

[3] The size of the main memory is typically denoted by M, but we use W to distinguish from the relation size in the previous sections.

[4] Even if this information is not available initially to the algorithm, we can easily obtain it by performing a single pass over the input data, which will cost $O(|I|/B)$ I/Os.

a rigorous way. We should also note that previous work [9] has studied the simulation of MapReduce algorithms on a parallel external memory model.

Let $\mathcal{A}$ be a tuple-based MPC algorithm that computes query q over input I using r rounds with load $L(I,p)$. We show next how to construct an external memory algorithm $\mathcal{B}$ based on the algorithm $\mathcal{A}$.

Simulation. The external memory algorithm $\mathcal{B}$ simulates the computation of algorithm $\mathcal{A}$ during each of the r rounds: round k, for $k = 1, \ldots, r$ simulates the total computation of the p servers during round k of $\mathcal{A}$. We pick a parameter p for the number of servers that we show how to compute later. The algorithm will store tuples of the form (t, s) to denote that tuple t resides in server s.

To initialize $\mathcal{B}$, we first assign the input data to the p servers (we can do this in any arbitrary way, as long as the data is equally distributed). More precisely, we read each tuple t of the input relations and then produce a tuple (t, s), where $s = 1, \ldots, p$ in a round-robin fashion, such that in the end each server is assigned $|I|/B$ data items. To achieve this, we load each relation in chunks of size B in the memory. After the initialization, the algorithm $\mathcal{B}$, for each round $k = 1, \ldots, r$, performs the following steps:

1. All tuples, which will be of the form (t, s), are sorted according to the attribute s.
2. All tuples are loaded in memory in chunks of size W, in the order by which they were sorted in the external memory. If we choose p such that $r \cdot L(I,p) \leq W$, we can fit in the internal memory all the tuples of any server s at round k. [5] Hence, we first read into the internal memory the tuples for server 1, then server 2, and so on. For each server s, we replicate in the internal memory the execution of algorithm $\mathcal{A}$ in server s at round k.
3. For each tuple t in server s (including the ones that are newly produced), we compute the tuples $\{(t, s') \mid s' \in \mathcal{D}(t,k)\}$, and we write them into the external memory in blocks of size B.

In other words, writing to the internal and external memory simulates the communication step, where data is exchanged between servers. The algorithm $\mathcal{B}$ produces the correct result, since by the choice of p we guarantee that we can load enough data in the memory to simulate the local computation of $\mathcal{A}$ at each server. Observe that we do not need to write the final result back to the external memory, since at the end of the last round we can just call $emit()$ for each tuple in the output.

Let us now identify the choice for p; recall that we must make sure that $r \cdot L(I,p) \leq W$. Hence, we must choose p_o such that $p_o = \min_p\{L(I,p) \leq W/r\}$. We next analyze the I/O cost of algorithm $\mathcal{B}$ for this choice of p_o.

Analysis. The initialization I/O cost for the algorithm is $|I|/B$. To analyze the cost for a given round $k = 1, \ldots, r$, we will measure first the size of the data that will be sorted and then loaded into memory at round k. For this, observe that at every round of algorithm $\mathcal{B}$, the total amount of data that is communicated is at most $p_o \cdot L(I,p_o)$. Hence, the total amount of data that will be loaded into memory will be at most $k \cdot p_o \cdot L(I,p_o) \leq p_o W$, from our definition of p_o.

For the first step that requires sorting the data, we will not use a sorting algorithm, but instead we will partition the data into p parts, and then concatenate the parts (this is

[5] The quantity $L(I,p)$ measures the maximum amount of data received during any round. Since data is not destroyed, over r rounds a server can receive as much as $r \cdot L(I,p)$ data. All of this data must fit into the memory of size W, since the decisions of each server depend on all the data received.

possible only if p_o is smaller than the memory W, i.e. it must be $p_o \leq W$). We can do this with a cost of $O(p_o W/B)$ I/Os. The second step of loading the tuples into memory has a cost of $p_o W/B$, since we are loading the data using chunks of size B; we can do this since the data has been sorted according to the destination server. As for the third step of writing the data into the external memory, observe that the total number of tuples written will be equal to the number of tuples communicated to the servers at round $k+1$, which will be at most $p_o L(I, p_o) \leq p_o W/r$. Hence, the I/O cost will be $p_o W/(rB)$.

Summing the I/O cost of all three steps over r rounds, we obtain that the I/O cost of the constructed algorithm $\mathcal{B}$ will be:

$$O\left(\frac{|I|}{B} + \sum_{k=1}^{r}\left(\frac{p_o W}{B} + \frac{p_o W}{rB}\right)\right) = O\left(\frac{|I|}{B} + \frac{r p_o W}{B}\right)$$

We have thus proved the following theorem:

▶ **Theorem 17.** *Let $\mathcal{A}$ be a tuple-based MPC algorithm that computes query q over input I using r rounds with load $L(I,p)$. For internal memory size W, let $p_o = \min_p\{L(I,p) \leq W/r\}$. If $W \geq p_o$, then there exists an external memory algorithm $\mathcal{B}$ that computes q over the same input I with I/O cost:*

$$O\left(\frac{|I|}{B} + \frac{r p_o W}{B}\right).$$

We can simplify the above I/O cost further in the context of computing conjunctive queries. In all of our algorithms we used a constant number of rounds r, and the load is typically $L(I,p) \geq |I|/p$. Then, we can rewrite the I/O cost as $O\left(p_o W/B\right)$.

We can apply Theorem 17 to any of the optimal multi-round algorithms we presented in the previous sections, and obtain state-of-the-art external memory algorithms for several classes of conjunctive queries. We show next an application for the case of query C_3.

▶ **Example 18.** We presented a 2-round algorithm that computes triangles for any input data with load (in tuples) $L = \tilde{O}(m/p^{3/2})$, in the case where all relations have size m. By applying Theorem 17, we obtain an external memory algorithm that computes triangles with $\tilde{O}(m^{3/2}/(BW^{1/2}))$ I/O cost for any $W \geq m^{2/5}$. Notice that this cost matches the I/O cost for triangle computation from [18] up to polylogarithmic factors.

6 Conclusion

In this work, we present the first worst-case analysis for parallel algorithms that compute conjunctive queries, using the MPC model as the theoretical framework for the analysis. We also show an interesting connection with the external memory computation model, which allows us to translate many of the techniques from the parallel setting to obtain algorithms for conjunctive queries with (almost) optimal I/O cost.

The central remaining open question is to design worst-case optimal algorithms for multiple rounds for any conjunctive query. We also plan to investigate further the connection between the parallel setting and external memory setting. It is an interesting question whether our techniques can lead to optimal external memory algorithms for any conjunctive query, and also whether we can achieve a reverse simulation of external memory algorithms in the MPC model.

Acknowledgements. We would like to thank Ke Yi for pointing out an error in the computation of the edge quasi-packing of the query L_k.

References

1 Foto N. Afrati, Anish Das Sarma, Semih Salihoglu, and Jeffrey D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *CoRR*, abs/1206.4377, 2012.

2 Foto N. Afrati and Jeffrey D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, pages 99–110, 2010. doi:10.1145/1739041.1739056.

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

4 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *PODS*, pages 273–284, 2013. doi:10.1145/2463664.2465224.

5 Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In *PODS*, pages 212–223, 2014. doi:10.1145/2594538.2594558.

6 Shumo Chu and James Cheng. Triangle listing in massive networks. *TKDD*, 6(4):17, 2012. doi:10.1145/2382577.2382581.

7 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

8 Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Clifford Stein, and Zoya Svitkina. On distributing symmetric streaming computations. *ACM Transactions on Algorithms*, 6(4), 2010.

9 Gero Greiner and Riko Jacob. The efficiency of mapreduce in parallel external memory. In *Proceedings of the 10th Latin American International Conference on Theoretical Informatics*, LATIN'12, pages 433–445, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-29344-3_37.

10 Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu. Demonstration of the Myria big data management service. 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 881–884. ACM, 2014. doi:10.1145/2588555.2594530.

11 Xiaocheng Hu, Miao Qiao, and Yufei Tao. Join dependency testing, Loomis-Whitney join, and triangle enumeration. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 – June 4, 2015*, pages 291–301, 2015. doi:10.1145/2745754.2745768.

12 Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. Massive graph triangulation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 325–336, 2013. doi:10.1145/2463676.2463704.

13 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA*, pages 938–948, 2010.

14 Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'15, pages 391–410. SIAM, 2015.

15 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *PODS*, pages 223–234, 2011. doi:10.1145/1989284.1989310.

16 Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. *PVLDB*, 3(1):330–339, 2010.

17 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *PODS*, pages 37–48, 2012. doi:10.1145/2213556.2213565.

18 Rasmus Pagh and Francesco Silvestri. The input/output complexity of triangle enumeration. In *PODS*, pages 224–233, 2014. doi:10.1145/2594538.2594552.

19 Francesco Silvestri. Subgraph enumeration in massive graphs. *CoRR*, abs/1402.3444, 2014. URL: http://arxiv.org/abs/1402.3444.

20 DavidP. Woodruff and Qin Zhang. When distributed computation is communication expensive. In Yehuda Afek, editor, *Distributed Computing*, volume 8205 of *Lecture Notes in Computer Science*, pages 16–30. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-41527-2_2.

21 M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

Parallel-Correctness and Containment for Conjunctive Queries with Union and Negation

Gaetano Geck[1], Bas Ketsman*[2], Frank Neven[3], and Thomas Schwentick[4]

1 **TU Dortmund University, Dortmund, Germany**
2 **Hasselt University, Hasselt, Belgium; and Transnational University of Limburg, Belgium/The Netherlands**
3 **Hasselt University, Hasselt, Belgium; and Transnational University of Limburg, Belgium/The Netherlands**
4 **TU Dortmund University, Dortmund, Germany**

Abstract

Single-round multiway join algorithms first reshuffle data over many servers and then evaluate the query at hand in a parallel and communication-free way. A key question is whether a given distribution policy for the reshuffle is adequate for computing a given query, also referred to as parallel-correctness. This paper extends the study of the complexity of parallel-correctness and its constituents, parallel-soundness and parallel-completeness, to unions of conjunctive queries with and without negation. As a by-product it is shown that the containment problem for conjunctive queries with negation is coNEXPTIME-complete.

1998 ACM Subject Classification H.2.3 Query Languages, H.2.4 Distributed databases

Keywords and phrases Conjunctive queries, distributed evaluation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.9

1 Introduction

Motivated by recent in-memory systems like Spark [7] and Shark [21], Koutris and Suciu introduced the massively parallel communication model (MPC) [15] where computation proceeds in a sequence of parallel steps each followed by global synchronisation of all servers. Of particular interest in the MPC model are queries that can be evaluated in one round of communication [9]. In its most naïve setting, a query $\mathcal{Q}$ is evaluated by reshuffling the data over many servers, according to some distribution policy, and then computing $\mathcal{Q}$ at each server in a parallel but communication-free manner. A notable family of distribution policies is formed within the Hypercube algorithm [3, 9, 11]. A property of Hypercube distributions is that for any instance I, the central execution of $\mathcal{Q}(I)$ always equals the union of the evaluations of $\mathcal{Q}$ at every computing node (or server). The latter guarantees the correctness of the distributed evaluation for any conjunctive query by the Hypercube algorithm.

Ameloot et al. [4] introduced a general framework for reasoning about one-round evaluation algorithms under *arbitrary* distribution policies. They introduced *parallel-correctness* as a property of a query w.r.t. a distribution policy which states that central execution always equals distributed execution, that is, equals the union of the evaluations of the query at each server under the given distribution policy. One of the main results of [4] is that deciding

* PhD Fellow of the Research Foundation – Flanders (FWO).

parallel-correctness for conjunctive queries (CQs) is Π_2^P-complete under arbitrary distribution policies. The upper bound follows rather directly from a semantical characterisation of parallel-correctness in terms of properties of minimal valuations. Specifically, it was shown that a conjunctive query is parallel-correct w.r.t. a distribution policy, if the distribution policy sends for every minimal valuation its required facts to at least one node.

As union and negation are fundamental operators, we extend in this paper the study of parallel-correctness to unions of conjunctive queries (UCQ), conjunctive queries with negation ($\text{CQ}^\neg$) and unions of conjunctive queries with negation ($\text{UCQ}^\neg$). In fact, we study two additional but related notions: parallel-soundness and parallel-completeness. While parallel-correctness implies equivalence between centralised and distributed execution, parallel-soundness (respectively, parallel-completeness) requires that distributed execution is contained in (respectively, contains) centralised execution. Of course, parallel-soundness and parallel-completeness together are equivalent to parallel-correctness. Furthermore, since all monotone queries are parallel-sound, on this class parallel-correctness is equivalent to parallel-completeness.

We start by investigating parallel-correctness for UCQ. Interestingly, for a UCQ to be parallel-correct under a certain distribution policy it is not required that every disjunct is parallel-correct. We extend the characterisation for parallel-correctness in terms of minimal valuations for CQs to UCQs and thereby obtain membership in Π_2^P. The matching lower bound follows, of course, from the lower bound for CQs [4].

Next, we study parallel-correctness for (unions of) conjunctive queries with negation. Sadly, when negation comes into play, parallel-correctness can no longer be characterised in terms of properties of valuations. Instead our algorithms are based on counter-examples of exponential size, yielding coNEXPTIME upper bounds. It turns out that this is optimal, though, as our corresponding lower bounds show. The proof of the lower bounds comes along an unexpected route: we exhibit a reduction from query containment for $\text{CQ}^\neg$ to parallel-correctness of $\text{CQ}^\neg$ (and its two variants) and show that query containment for $\text{CQ}^\neg$ is coNEXPTIME-complete. This is considerably different from what we thought was folklore knowledge of the community. Indeed, the Π_2^p-completeness result for query containment for $\text{CQ}^\neg$ mentioned in [19] only seems to hold for fixed database schemas (or a fixed arity bound, for that matter). We note that Mugnier et al. [17] provide a Π_2^p upper bound proof for $\text{CQ}^\neg$ containment and explicitly mention that it holds under the assumption that the arity of predicates is bounded by a constant. Altogether, parallel-correctness (and its variants) for (unions of) conjunctive queries with negation is thus complete for coNEXPTIME.

Finally, a natural question is how the high complexity of parallel-correctness in the presence of negation can be lowered. We identify two cases in which the complexity drops. More specifically, the complexity decreases from coNEXPTIME to Π_2^p if the database schema is fixed or the arity of relations is bounded, and to coNP for unions of *full* conjunctive queries with negation. In the latter case, we again employ a reduction from containment of full conjunctive queries (with negation) and obtain novel results on the containment problem in this setting as well. All upper bounds hold for queries with inequalities.

Outline. This paper is further organised as follows. In Section 2, we discuss related work. In Section 3, we introduce the necessary definitions. We address parallel-correctness for unions of conjunctive queries in Section 4. We consider containment of conjunctive queries with negation in Section 5 and parallel-correctness together with its variants in Section 6. We discuss the restriction to full conjunctive queries in Section 7. We conclude in Section 8.

Missing proof details can be found in the full version of this paper [14].

2 Related work

As mentioned in the introduction, Koutris and Suciu introduced the massively parallel communication model (MPC) [15]. A key property is that computation proceeds in a sequence of parallel steps, each followed by global synchronisation of all computing nodes. In this model, evaluation of conjunctive queries [8, 15] and skyline queries [2] has been considered. Beame, Koutris and Suciu [9] proved a matching upper and lower bound for the amount of communication needed to compute a full conjunctive query without self-joins in one communication round. The upper bound is provided by a randomised algorithm called *Hypercube* which uses a technique that can be traced back to Ganguly, Silberschatz, and Tsur [13] and is described in the context of map-reduce by Afrati and Ullman [3].

Ameloot et al. [4] introduced a general framework for reasoning about one-round evaluation algorithms under *arbitrary* distribution policies. They introduced the notion of *parallel-correctness* and proved its associated decision problem to be Π_2^p-complete for conjunctive queries. In addition, towards optimisation in MPC, they considered parallel-correctness *transfer*. Here, parallel-correctness transfers from $\mathcal{Q}$ to $\mathcal{Q}'$ when $\mathcal{Q}'$ is parallel-correct under every distribution policy for which $\mathcal{Q}$ is parallel-correct. The associated decision problem for conjunctive queries is shown to be Π_3^p-complete. In addition, some restricted cases (e.g., transferability under Hypercube distributions), are shown to be NP-complete.

Our definition of a distribution policy is borrowed from Ameloot et al. [5] (but already surfaces in the work of Zinn et al. [22]), where distribution policies are used to define the class of policy-aware transducer networks. The work by Ameloot et al. [6, 5] relates coordination-free computation with definability in variants of Datalog. One-round communication algorithms in MPC can be seen as very restrictive coordination-free computation.

The complexity of query containment for conjunctive queries is proved to be NP-complete by Chandra and Merlin [10]. Levy and Sagiv provide a test for query containment of conjunctive queries with negation [16] that involves exploring an exponential number of possible counter-example instances. In the context of information integration, Ullman [19] gives a comprehensive overview of query containment (with and without negation) and states the complexity of query containment for $\text{CQ}^\neg$ to be Π_2^p-complete. As mentioned in the introduction, the latter apparently only holds when the database schema is fixed or the arity of relations is considered to be bounded. A proof for the Π_2^p-lowerbound is given by Farré et al. [12]. Based on [16], Wei and Lausen [20] study a method for testing containment that exploits containment mappings for the positive parts of queries, and additionally provide a characterisation for $\text{UCQ}^\neg$ containment.

3 Definitions

3.1 Queries and instances

We assume an infinite set **dom** of data values that can be represented by strings over some fixed alphabet. By $\mathbf{dom}_n$ we denote the set of data values represented by strings of length at most n. A *database schema* $\mathcal{D}$ is a finite set of relation names R, each with some arity $ar(R)$. We also write $R^{(k)}$ as a shorthand to denote that R is a relation of arity k. We call $R(\mathbf{t})$ a *fact* when R is a relation name and $\mathbf{t}$ a tuple over **dom** of appropriate arity. We say that a fact $R(\mathbf{t})$ is *over* a database schema $\mathcal{D}$ if $R \in \mathcal{D}$. For a subset $U \subseteq \mathbf{dom}$ we write $facts(\mathcal{D}, U)$ for the set of possible facts over schema $\mathcal{D}$ and U and by $facts(\mathcal{D})$ we denote $facts(\mathcal{D}, \mathbf{dom})$. A *(database) instance* I over $\mathcal{D}$ is a finite set of facts over $\mathcal{D}$. By $adom(I)$ we denote the set of data values occurring in I. A *query* $\mathcal{Q}$ *over input schema* $\mathcal{D}_1$ *and output*

schema $\mathcal{D}_2$ is a generic mapping from instances over $\mathcal{D}_1$ to instances over $\mathcal{D}_2$. Genericity means that for every permutation π of **dom** and every instance I, $\mathcal{Q}(\pi(I)) = \pi(\mathcal{Q}(I))$. We say that $\mathcal{Q}$ is *contained* in $\mathcal{Q}'$, denoted $\mathcal{Q} \subseteq \mathcal{Q}'$ iff for all instances I, $\mathcal{Q}(I) \subseteq \mathcal{Q}'(I)$.

3.2 Unions of conjunctive queries with negation

Let **var** be an infinite set of variables, disjoint from **dom**. An *atom* over schema $\mathcal{D}$ is of the form $R(\mathbf{x})$, where R is a relation name from $\mathcal{D}$ and $\mathbf{x} = (x_1, \ldots, x_k)$ is a tuple of variables in **var** with $k = ar(R)$. A *conjunctive query* $\mathcal{Q}$ *with negation and inequalities* over input schema $\mathcal{D}$ is an expression of the form

$$T(\mathbf{x}) \leftarrow R_1(\mathbf{y_1}), \ldots, R_m(\mathbf{y_m}), \neg S_1(\mathbf{z_1}), \ldots, \neg S_n(\mathbf{z_n}), \beta_1, \ldots, \beta_p$$

where all $R_i(\mathbf{y}_i)$ and $S_i(\mathbf{z}_j)$ are atoms over $\mathcal{D}$, every β_i is an inequality of the form $s \neq s'$ where s, s' are distinct variables occurring in some $\mathbf{y}_i$ or $\mathbf{z}_j$, and $T(\mathbf{x})$ is an atom for which $T \notin \mathcal{D}$. Additionally, for safety, we require that every variable in $\mathbf{x}$ occurs in some $\mathbf{y}_i$ and that every variable occurring in a negated atom has to occur in a positive atom as well (*safe* negation). We refer to the *head atom* $T(\mathbf{x})$ as $head_{\mathcal{Q}}$, to the set $\{R_1(\mathbf{y_1}), \ldots, R_m(\mathbf{y_m}), S_1(\mathbf{z_1}), \ldots, S_n(\mathbf{z_n})\}$ as $body_{\mathcal{Q}}$, and to the set $\{\beta_1, \ldots, \beta_p\}$ as $ineq_{\mathcal{Q}}$. Specifically, we refer to $\{R_1(\mathbf{y_1}), \ldots, R_m(\mathbf{y_m})\}$ as the positive atoms in $\mathcal{Q}$, denoted $pos_{\mathcal{Q}}$, and to $\{S_1(\mathbf{z_1}), \ldots, S_n(\mathbf{z_n})\}$ as the *negated* atoms of $\mathcal{Q}$, denoted $neg_{\mathcal{Q}}$. We denote by $vars(\mathcal{Q})$ the set of all variables occurring in $\mathcal{Q}$. We refer to the class of conjunctive queries with negation and inequalities by $\mathbf{CQ}^{\neg,\neq}$, its restriction to queries without inequalities, without negated atoms, and without both by $\mathbf{CQ}^{\neg}$, $\mathbf{CQ}^{\neq}$, and $\mathbf{CQ}$, respectively. As a shorthand we refer to queries from $\mathbf{CQ}^{\neg,\neq}$ as $\mathrm{CQ}^{\neg,\neq}$s and similarly for the other classes.

A *pre-valuation* for a $\mathrm{CQ}^{\neg,\neq}$ $\mathcal{Q}$ is a total function $V : vars(\mathcal{Q}) \to \mathbf{dom}$, which naturally extends to atoms and sets of atoms. It is *consistent* for $\mathcal{Q}$, if $V(pos_{\mathcal{Q}}) \cap V(neg_{\mathcal{Q}}) = \emptyset$, and $V(s) \neq V(s')$, for every inequality $s \neq s'$ of $\mathcal{Q}$, in which case it is called a valuation. Of course, for a conjunctive query without negated atoms and without inequalities, every pre-valuation is also a valuation. We refer to $V(pos_{\mathcal{Q}})$ as the facts *required* by V, and to $V(neg_{\mathcal{Q}})$ as the facts *prohibited* by V.

A valuation V *satisfies* $\mathcal{Q}$ on instance I if all facts required by V are in I while no fact prohibited by V is in I, that is, if $V(pos_{\mathcal{Q}}) \subseteq I$ and $V(neg_{\mathcal{Q}}) \cap I = \emptyset$. In that case, V *derives* the fact $V(head_{\mathcal{Q}})$. The *result of* $\mathcal{Q}$ *on instance* I, denoted $\mathcal{Q}(I)$, is defined as the set of facts that can be derived by satisfying valuations for $\mathcal{Q}$ on I.

A *union of conjunctive queries with negation and inequalities* is a finite union of $\mathrm{CQ}^{\neg,\neq}$s. That is, $\mathcal{Q}$ is of the form $\bigcup_{i=1}^{n} \mathcal{Q}_i$ where all subqueries $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$ have the same relation name in their head atoms. We assume disjoint variable sets among different disjuncts in $\mathcal{Q}$. That is, $vars(\mathcal{Q}_i) \cap vars(\mathcal{Q}_j) = \emptyset$ for $i \neq j$ and, in particular, $vars(head_{\mathcal{Q}_i}) \neq vars(head_{\mathcal{Q}_j})$. By $varmax(\mathcal{Q})$ we denote the maximum number of variables that occurs in any disjunct of $\mathcal{Q}$. By $\mathbf{UCQ}^{\neg,\neq}$ we denote the class of unions of conjunctive queries with negation and inequalities and its fragments are denoted correspondingly.

A $\mathrm{CQ}^{\neg,\neq}$ is called *full* if all of its variables occur in its head. A $\mathrm{UCQ}^{\neg,\neq}$ is *full* if all its subqueries are full.

The *result of* $\mathcal{Q}$ *on instance* I is $\mathcal{Q}(I) = \bigcup_{i=1}^{n} \mathcal{Q}_i(I)$. Accordingly, a mapping from variables to data values is a *valuation* for a $\mathrm{UCQ}^{\neg,\neq}$ $\mathcal{Q}$ if it is a valuation for one of its subqueries.

3.3 Networks, data distribution, and policies

A *network* $\mathcal{N}$ is a nonempty finite set of values from **dom**, which we call *(computing) nodes* (or servers). A *distribution policy* $\boldsymbol{P} = (U, rfacts_{\boldsymbol{P}})$ for a database schema $\mathcal{D}$ and a network $\mathcal{N}$ consists of a universe U and a total function $rfacts_{\boldsymbol{P}}$ that maps each node of $\mathcal{N}$ to a set of facts from $facts(\mathcal{D}, U)$. A node κ is *responsible for fact* $\boldsymbol{f}$ (under policy $\boldsymbol{P}$) if $\boldsymbol{f} \in rfacts_{\boldsymbol{P}}(\kappa)$. As a shorthand (and slight abuse of notation), we denote the set of nodes κ that are responsible for some given fact $\boldsymbol{f}$ by $\boldsymbol{P}(\boldsymbol{f})$. For a distribution policy $\boldsymbol{P}$ and an instance I over $\mathcal{D}$, let $loc\text{-}inst_{\boldsymbol{P},I}$ denote the function that maps each $\kappa \in \mathcal{N}$ to $I \cap rfacts_{\boldsymbol{P}}(\kappa)$, that is, the set of facts in I for which κ is responsible. We sometimes refer to a given instance I as the *global instance* and to $loc\text{-}inst_{\boldsymbol{P},I}(\kappa)$ as the *local instance at node* κ.

We note that for some facts from $facts(\mathcal{D}, U)$ there are no responsible nodes. This gives our framework some additional flexibility. However, it does not affect our results: in the lower bound proofs we only use distributions for which all facts from $facts(\mathcal{D}, U)$ have some responsible nodes. Each distribution policy implicitly induces a network and each query implicitly defines a database (sub-) schema. Therefore, we often omit the explicit notation for networks and schemas.

Given some policy $\boldsymbol{P}$ that is defined over a network $\mathcal{N}$, the *result* $[\mathcal{Q}, \boldsymbol{P}](I)$ *of the distributed evaluation of a query* $\mathcal{Q}$ *on an instance* I *in one round* is defined as the union of the results of the query evaluated on each node's local instance. Formally,

$$[\mathcal{Q}, \boldsymbol{P}](I) \stackrel{\text{def}}{=} \bigcup_{\kappa \in \mathcal{N}} \mathcal{Q}\big(loc\text{-}inst_{\boldsymbol{P},I}(\kappa)\big).$$

In the decision problem for parallel correctness (to be formalised later), the input consists of a query $\mathcal{Q}$ and a distribution policy $\boldsymbol{P}$. However, it is not obvious how distribution policies should be specified. In principle, they could be defined in an arbitrary fashion, but it is reasonable to assume that given a potential fact $\boldsymbol{f}$, a node κ and a policy $\boldsymbol{P}$, it is not too hard to find out whether κ is responsible for $\boldsymbol{f}$ under $\boldsymbol{P}$.

For UCQ$^{\neq}$s, which are monotone, our complexity results are remarkably robust with respect to the choice of the representation of distribution policies. In fact, the complexity results coincide for the two extreme possible choices that we consider in this article. In the first case, distribution policies are specified by an explicit list of tuple-node-pairs, whereas in the second case the test whether a given node is responsible for a given tuple can be carried out by a non-deterministic polynomial-time algorithm. However, we do require that some bound n on the length of strings that represent node names and data values is given. Without such a restriction, no upper complexity bounds would be possible as nodes with names of super-polynomial length in the size of the input would not be accessible.

Considering queries with negated atoms, however, these two settings (seem to) differ, complexity-wise. The reason is that testing parallel-correctness in this setting requires counter examples of size exponential in the size of the query which can not be succinctly represented by policies in $\mathcal{P}_{\text{fin}}$. We therefore introduce the class $\mathcal{P}_{\text{rule}}$ allowing for a more economic rule based description of policies. In particular, in $\mathcal{P}_{\text{rule}}$, the universe U of a policy is explicitly enumerated and the responsibilities are defined by simple constraints (described below). The latter representation enjoys the same complexity properties as the full NP-test based case.

Now we give more precise definitions of classes of policies and their representations as inputs of algorithmic problems. As said before, policies $\boldsymbol{P} = (U, rfacts_{\boldsymbol{P}})$ from $\mathcal{P}_{\text{fin}}$ are specified by an explicit enumeration of U and of all pairs $(\kappa, \boldsymbol{f})$ where $\kappa \in \boldsymbol{P}(\boldsymbol{f})$. A policy $\boldsymbol{P} = (U, rfacts_{\boldsymbol{P}})$ from $\mathcal{P}_{\text{rule}}$ is given by an explicit enumeration of U and a list of *rules* of the form $\rho = (A, \kappa)$, where A is an atom with variables and/or constants from U,

and a network node κ. The semantics of such a rule is as follows: for every substitution $\mu : \mathbf{var} \cup \mathbf{dom} \to \mathbf{dom}$ that maps variables to values from U and leaves constants from U unchanged, the node κ is responsible for the fact $\mu(A)$. A rule is a *fact rule* if its atom does not contain any variables, that is, $A = R(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n \in U$. In particular, $\mathcal{P}_{\text{fin}} \subseteq \mathcal{P}_{\text{rule}}$.

▶ **Example 1.** Let distribution policy $\boldsymbol{P}$ over schema $\{\texttt{Rel}^{(3)}\}$ and network $\{\kappa_1, \kappa_2\}$ be given by $U = \{1, \ldots, 10\}$ and the rules $\big(\texttt{Rel}(1, x, x), \kappa_1\big), \big(\texttt{Rel}(2, x, y), \kappa_2\big)$. On global instance $I = \{\texttt{Rel}(1,7,7), \texttt{Rel}(1,7,8), \texttt{Rel}(2,9,8), \texttt{Rel}(2,9,9)\}$, policy $\boldsymbol{P}$ induces local instances $\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa_1) = \{\texttt{Rel}(1,7,7)\}$ and $\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa_2) = \{\texttt{Rel}(2,9,8), \texttt{Rel}(2,9,9)\}$. □

The most general classes of policies allow to specify policies by means of a 'test algorithm' with time bound ℓ^k, where ℓ is the length of the input and k some constant. Such an algorithm decides, for an input consisting of a node κ and fact $\boldsymbol{f}$, whether κ is responsible for $\boldsymbol{f}$.[1] A policy $\boldsymbol{P} = (U, \mathit{rfacts}_{\boldsymbol{P}})$ from $\mathcal{P}^k_{\text{npoly}}$ is specified by a pair $(n, \mathcal{A}_{\boldsymbol{P}})$, where n is a natural number in unary representation and $\mathcal{A}_{\boldsymbol{P}}$ is a non-deterministic algorithm.[2] The universe U of $\boldsymbol{P}$ is the set of all data values that can be represented by strings of length at most n (for some given fixed alphabet) and the underlying network consists of all nodes which are represented by strings of length at most n, that is, $\mathcal{N} = \mathbf{dom}_n$. A node κ is responsible for a fact $\boldsymbol{f}$ if $\mathcal{A}_{\boldsymbol{P}}$, on input $(\kappa, \boldsymbol{f})$, has an accepting run of at most $|(\kappa, \boldsymbol{f})|^k$ steps. Clearly, each policy of $\mathcal{P}_{\text{fin}}$ can be described in $\mathcal{P}^2_{\text{npoly}}$. Let $\mathfrak{P}_{\text{npoly}}$ denote the set[3] $\{\mathcal{P}^k_{\text{npoly}} \mid k \geq 2\}$ of distribution policies and by $\mathfrak{P}$ the set $\{\mathcal{P}_{\text{fin}}, \mathcal{P}_{\text{rule}}\} \cup \mathfrak{P}_{\text{npoly}}$.

3.4 Parallel-correctness, soundness, and completeness

In this paper, we mainly consider the one-round evaluation algorithm for a query $\mathcal{Q}$ that first distributes (reshuffles) the data over the computing nodes according to $\boldsymbol{P}$, then evaluates Q in a parallel step at every computing node, and finally outputs all facts that are obtained in this way.[4] As formalised next, the one-round evaluation algorithm is correct (sound, complete) if the query $\mathcal{Q}$ is parallel-correct (parallel-sound, parallel-complete) under $\boldsymbol{P}$.

▶ **Definition 2.** Let $\mathcal{Q}$ be a query, I an instance, and $\boldsymbol{P}$ a distribution policy.

- $\mathcal{Q}$ is *parallel-sound on I under $\boldsymbol{P}$* if $\mathcal{Q}(I) \supseteq [\mathcal{Q}, \boldsymbol{P}](I)$.
- $\mathcal{Q}$ is *parallel-complete on I under $\boldsymbol{P}$* if $\mathcal{Q}(I) \subseteq [\mathcal{Q}, \boldsymbol{P}](I)$; and,
- $\mathcal{Q}$ is *parallel-correct on I under $\boldsymbol{P}$* if $\mathcal{Q}(I) = [\mathcal{Q}, \boldsymbol{P}](I)$, that is, if it is parallel-sound and parallel-complete.

▶ **Definition 3.** A query $\mathcal{Q}$ is *parallel-correct (respectively, parallel-sound and parallel-complete) under distribution policy $\boldsymbol{P} = (U, \mathit{rfacts}_{\boldsymbol{P}})$*, if $\mathcal{Q}$ is parallel-correct (respectively, parallel-sound and parallel-complete) on all instances $I \subseteq \mathit{facts}(\mathcal{D}, U)$.

In [4], parallel-correctness is characterised in terms of minimal valuations as defined next:

▶ **Definition 4.** Let $\mathcal{Q}$ be a CQ. A valuation V for $\mathcal{Q}$ is *minimal* for $\mathcal{Q}$ if there exists no valuation V' for $\mathcal{Q}$ such that $V(\mathit{head}_{\mathcal{Q}}) = V'(\mathit{head}_{\mathcal{Q}})$ and $V'(\mathit{body}_{\mathcal{Q}}) \subsetneq V(\mathit{body}_{\mathcal{Q}})$.

---

[1] We note that it is important that for each class of policies there is a fixed k that bounds the exponent in the test algorithm as otherwise we could not expect a polynomial bound for all policies of that class.
[2] For concreteness, say, a non-deterministic Turing machine.
[3] Since 'linear time' is a subtle notion, we rather not consider $\mathcal{P}^1_{\text{npoly}}$.
[4] We note that, since $\boldsymbol{P}$ is defined on the granularity of a fact, the reshuffling does not depend on the current distribution of the data and can be done in parallel as well.

The following lemma is key in obtaining the Π_2^p upper bound on the complexity of testing parallel-correctness for conjunctive queries:

▶ **Lemma 5** (Characterisation of parallel-correctness for CQs [4]). *A CQ $\mathcal{Q}$ is parallel-correct under distribution policy $\boldsymbol{P} = (U, rfacts_{\boldsymbol{P}})$ if and only if the following holds:*

For every minimal valuation V for $\mathcal{Q}$ over U, there is a node $\kappa \in \mathcal{N}$ such that $V(body_{\mathcal{Q}}) \subseteq rfacts_{\boldsymbol{P}}(\kappa)$. (*C1*)

▶ **Remark 6.** *Informally, condition (C1) states that there is a node in the network where all facts required for V meet.*

3.5 Algorithmic problems

We consider the following decision problems for various sub-classes $\mathcal{C}$ and $\mathcal{C}'$ of $\mathbf{UCQ}^{\neg,\neq}$ and classes $\mathcal{P}$ of distribution policies from $\{\mathcal{P}_{\text{fin}}, \mathcal{P}_{\text{rule}}\} \cup \mathfrak{P}_{\text{npoly}}$.

Containment$(\mathcal{C}, \mathcal{C}')$:	Parallel-Sound$(\mathcal{C}, \mathcal{P})$:
Input: $\mathcal{Q} \in \mathcal{C}$ and $\mathcal{Q}' \in \mathcal{C}'$	**Input:** $\mathcal{Q} \in \mathcal{C}$, $\boldsymbol{P} \in \mathcal{P}$
Question: Is $\mathcal{Q} \subseteq \mathcal{Q}'$?	**Question:** Is $\mathcal{Q}$ parallel-sound under $\boldsymbol{P}$?

Parallel-Complete$(\mathcal{C}, \mathcal{P})$:	Parallel-Correct$(\mathcal{C}, \mathcal{P})$:
Input: $\mathcal{Q} \in \mathcal{C}$, $\boldsymbol{P} \in \mathcal{P}$	**Input:** $\mathcal{Q} \in \mathcal{C}$, $\boldsymbol{P} \in \mathcal{P}$
Question: Is $\mathcal{Q}$ parallel-complete under $\boldsymbol{P}$?	**Question:** Is $\mathcal{Q}$ parallel-correct under $\boldsymbol{P}$?

4 Parallel-correctness: unions of conjunctive queries

Parallel-correctness of unions of conjunctive queries (without negation) reduces to parallel-completeness for the simple reason that these queries are monotone and therefore parallel-sound for every distribution policy. We show below that parallel-completeness remains in Π_2^p. Hardness already follows from Π_2^p-hardness of Parallel-Correct$(\mathbf{CQ}, \mathcal{P}_{\text{fin}})$ [4].

As a UCQ is parallel-complete under a policy $\boldsymbol{P}$ when all its disjuncts are, it might be tempting to assume that this condition is also necessary. However, as the following example illustrates, this is not the case.

▶ **Example 7.** Let $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$, where $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are the following CQs:

$$\begin{aligned} \mathcal{Q}_1 &: \quad H(x,x) \quad \leftarrow \quad R(x,x), \\ \mathcal{Q}_2 &: \quad H(y,z) \quad \leftarrow \quad R(y,z), S(y,z). \end{aligned}$$

Further, let $\boldsymbol{P}$ be the policy over network $\{\kappa_1, \kappa_2\}$ that maps facts $R(a,a)$ to node κ_1, for all $a \in \mathbf{dom}$, and all other R-facts and all S-facts to node κ_2.

We argue that $\mathcal{Q}$ is parallel-complete under $\boldsymbol{P}$ on all instances. Indeed, assume $H(a,b) \in \mathcal{Q}(I)$ for some instance I and $a, b \in \mathbf{dom}$. If $a \neq b$, only the valuation $\{y \mapsto a, z \mapsto b\}$ can derive $H(a,b)$. This means that $\{R(a,b), S(a,b)\} \subseteq I$. Furthermore, $\{R(a,b), S(a,b)\} \subseteq rfacts_{\boldsymbol{P}}(\kappa_2)$. Hence, $H(a,b) \in \mathcal{Q}(loc\text{-}inst_{\boldsymbol{P},I}(\kappa_2))$. If $a = b$, then $R(a,a) \in I$. So, $R(a,a) \in rfacts_{\boldsymbol{P}}(\kappa_1)$ and $H(a,a) \in \mathcal{Q}(loc\text{-}inst_{\boldsymbol{P},I}(\kappa_1))$. On the other hand, $\mathcal{Q}_2$ is not parallel-complete under $\boldsymbol{P}$ on instance $I = \{R(0,0), S(0,0)\}$. Indeed, $H(0,0) \in \mathcal{Q}_2(I)$ but $\mathcal{Q}_2\big(loc\text{-}inst_{\boldsymbol{P},I}(\kappa_1)\big) = \mathcal{Q}_2\big(\{R(0,0)\}\big) = \emptyset$ and $\mathcal{Q}_2\big(loc\text{-}inst_{\boldsymbol{P},I}(\kappa_2)\big) = \mathcal{Q}_2\big(\{S(0,0)\}\big) = \emptyset$. □

We recall from Section 3.2 that disjuncts in unions of conjunctive queries use disjoint variable sets and a valuation for $\mathcal{Q}$ is a valuation for exactly one disjunct. As formalised next, the notion of minimality for valuations given in Definition 4 naturally extends to $\mathbf{UCQ}^{\neq}$.

▶ **Definition 8.** Let $\mathcal{Q} = \bigcup_{i=1}^{n} \mathcal{Q}_i$ be a UCQ$^{\neq}$. A valuation V_i for $\mathcal{Q}_i$, with $i \in \{1, \ldots, n\}$, is *minimal* for $\mathcal{Q}$, if for no $j \in \{1, \ldots, n\}$ there is a valuation V_j for $\mathcal{Q}_j$, such that $V_j(\mathit{head}_{\mathcal{Q}_j}) = V_i(\mathit{head}_{\mathcal{Q}_i})$ and $V_j(\mathit{body}_{\mathcal{Q}_j}) \subsetneq V_i(\mathit{body}_{\mathcal{Q}_i})$.

▶ **Example 9.** Consider a simple UCQ$^{\neq}$ $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ where $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbf{CQ}^{\neq}$ are as follows:

$$\begin{array}{lllll} \mathcal{Q}_1: & H(u,v) & \leftarrow & R(u,v), R(v,u), R(u,u), \\ \mathcal{Q}_2: & H(x,y) & \leftarrow & R(x,y), R(y,z), y \neq z. \end{array}$$

Valuation $V_2 \stackrel{\text{def}}{=} \{x \mapsto 0, y \mapsto 0, z \mapsto 1\}$ is not minimal for $\mathcal{Q}$ because valuation $V_1 \stackrel{\text{def}}{=} \{u \mapsto 0, v \mapsto 0\}$ derives the same fact $H(0,0)$ requiring only $\{R(0,0)\} \subsetneq \{R(0,0), R(0,1)\}$. Similarly, valuation $W_1 \stackrel{\text{def}}{=} \{u \mapsto 0, v \mapsto 1\}$, requiring $\{R(0,1), R(1,0), R(0,0)\}$, is not minimal for $\mathcal{Q}$ because valuation $W_2 \stackrel{\text{def}}{=} \{x \mapsto 0, y \mapsto 1, z \mapsto 0\}$ only requires $\{R(0,1), R(1,0)\}$. □

The notion of minimality leads to basically the same simple characterisation of parallel-completeness:

▶ **Lemma 10.** *A UCQ$^{\neq}$ $\mathcal{Q}$ is parallel-correct under distribution policy $\boldsymbol{P} = (U, \mathit{rfacts}_{\boldsymbol{P}})$ if and only if the following holds:*

For every minimal valuation V for $\mathcal{Q}$ over U, there is a node $\kappa \in \mathcal{N}$ such that $V(\mathit{body}_{\mathcal{Q}}) \subseteq \mathit{rfacts}_{\boldsymbol{P}}(\kappa)$. ($C1'$)

Proof. *(If)* Assume ($C1'$) holds. Because of monotonicity, we only need to show that $\mathcal{Q}(I) \subseteq \bigcup_{\kappa \in \mathcal{N}} \mathcal{Q}(\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa))$ for every instance I. To this end, let $\boldsymbol{f}$ be an arbitrary fact that is derived by some valuation V for $\mathcal{Q}$ on I. Then, there is also a minimal valuation V' that is satisfying on I and which derives $\boldsymbol{f}$. Because of ($C1'$), there is a node $\kappa \in \mathcal{N}$ where all facts required by V' meet (cf. Remark 6). Hence, $\boldsymbol{f} \in \bigcup_{\kappa \in \mathcal{N}} \mathcal{Q}(\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa))$, i.e. query $\mathcal{Q}$ is parallel-correct under policy $\boldsymbol{P}$.

(Only if) For a proof by contraposition, suppose that there is a minimal valuation V' for $\mathcal{Q}$ for which the required facts do *not* meet under $\boldsymbol{P}$. Consider the input instance $I = V'(\mathit{body}_{\mathcal{Q}})$. By definition of minimality, there is no valuation that agrees on the head variables and is satisfying for $\mathcal{Q}$ on a strict subset of $V'(\mathit{body}_{\mathcal{Q}})$. Therefore, $V'(\mathit{head}_{\mathcal{Q}})$ is in $\mathcal{Q}(I)$ but it is not derived on any node and thus query $\mathcal{Q}$ is not parallel-complete under policy $\boldsymbol{P}$. ◀

The characterisation in Lemma 10, in turn, can be used to prove a Π_2^p upper bound.

▶ **Lemma 11.** Parallel-Correct($\boldsymbol{UCQ}^{\neq}, \mathcal{P}$) *is in Π_2^p, for every $\mathcal{P} \in \mathfrak{P}$.*

Proof. It suffices to show that the complement of Parallel-Complete($\mathbf{UCQ}^{\neq}, \mathcal{P}^k_{\text{npoly}}$) is in Σ_2^p for arbitrary $k \geq 2$. Let $\boldsymbol{P} = (n, T)$ be a policy from $\mathcal{P}^k_{\text{npoly}}$. We have to consider only instances whose data values can be represented by strings of length n over networks whose nodes can be represented by strings of length n.

By Lemma 10, a query $\mathcal{Q}$ is not parallel-correct under distribution policy $\boldsymbol{P}$ if and only if there exists a minimal valuation V that satisfies $\mathcal{Q}$ on some instance I with $\mathit{adom}(I) \subseteq \mathbf{dom}_n$ such that no node in $\mathbf{dom}_n$ is responsible for all facts from $V(\mathit{body}_{\mathcal{Q}})$.

First, the algorithm non-deterministically guesses a valuation V, which can be represented by a string in length polynomial in $\mathcal{Q}$ and n. Subsequently, it checks for all valuations V',

all nodes κ, and all strings x of polynomial length whether V' contradicts minimality of V (in which case the algorithm rejects the input) and, by use of algorithm T, whether node κ is not responsible for at least one fact from $V(body_Q)$ (if so, the algorithm continues, otherwise it rejects). All tests can be done in polynomial time. ◀

From [4] we know the following result.

▶ **Theorem 12** ([4]). Parallel-Correct($\boldsymbol{CQ}, \mathcal{P}_{fin}$) *is* Π_2^p*-complete.*

Together with Lemma 11 we get the following result.

▶ **Theorem 13.** Parallel-Correct($\boldsymbol{UCQ}^{\neq}, \mathcal{P}$) *is* Π_2^p*-complete, for every* $\mathcal{P} \in \mathfrak{P}$.

5 Containment of CQ$^\neg$ and UCQ$^\neg$

In this section, we establish the complexity of containment for $\mathbf{CQ}^\neg$ and $\mathbf{UCQ}^\neg$. We need these results to establish lower bounds on parallel-correctness and its constituents in the next section. Whereas containment for $\mathbf{CQ}$ has been intensively studied in the literature, the analogous problems for $\mathbf{CQ}^\neg$ and $\mathbf{UCQ}^\neg$ have hardly been addressed and seem to belong to folklore. In fact, we only found a reference of a complexity result for containment of $\mathbf{CQ}^\neg$ in [19], where a Π_2^p-algorithm for the problem is given, based on observations in [16], and the existence of a matching lower bound is mentioned. However, as we show below, although the problem is indeed in Π_2^p for queries defined over a fixed schema (or when the arity of relations is bounded), it is coNEXPTIME-complete in the general case.

We first show the lower bounds. They actually already hold for Boolean queries. We show that Containment($\mathbf{BCQ}^\neg, \mathbf{UBCQ}^\neg$) is coNEXPTIME-hard by a reduction from the succinct 3-colorability problem and afterwards that Containment($\mathbf{BCQ}^\neg, \mathbf{UBCQ}^\neg$) can be reduced to Containment($\mathbf{BCQ}^\neg, \mathbf{BCQ}^\neg$). Here, $\mathbf{BCQ}^\neg$ and $\mathbf{UBCQ}^\neg$ denote the class of Boolean CQ$^\neg$s and unions of Boolean CQ$^\neg$s, respectively. Together this establishes that Containment($\mathbf{BCQ}^\neg, \mathbf{BCQ}^\neg$) and therefore also Containment($\mathbf{CQ}^\neg, \mathbf{CQ}^\neg$) are coNEXPTIME-hard.

▶ **Proposition 14.** Containment($\boldsymbol{BCQ}^\neg$, $\boldsymbol{UBCQ}^\neg$) *is* coNEXPTIME*-hard.*

Proof. The proof is by a reduction from the succinct 3-colorability problem, which asks, whether a graph G, which is implicitly given by a circuit with binary AND- and OR- and unary NEG-gates, is 3-colorable. The latter problem is known to be NEXPTIME-complete [18]. We say that a circuit C, with 2ℓ Boolean inputs, describes a graph $G = (N, E)$, when $N = \{0,1\}^\ell$, and there is an edge $(n_1, n_2) \in N^2$ if and only if C outputs true on input $n_1 n_2$.

Let C be an input for the succinct 3-colorability problem with 2ℓ Boolean inputs. We construct queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that $\mathcal{Q}_1 \not\subseteq \mathcal{Q}_2$ if and only if the graph described by C is 3-colorable.

Both queries are over schema $\mathcal{D}$, which consists of relation names $\texttt{DomainValues}^{(3)}$, $\texttt{Bool}^{(1)}$, $\texttt{And}^{(3)}$, $\texttt{Or}^{(3)}$, $\texttt{Neg}^{(2)}$, and $\texttt{Label}^{(\ell+1)}$. Intuitively, satisfaction of $\mathcal{Q}_1$ will guarantee that there is a tuple (a_0, a_1, a_2) with three different values in relation $\texttt{DomainValues}$. We will use, for some such tuple, a_0, a_1, a_2 as colors and a_0, a_1 as truth values. We will often assume without loss of generality that $(a_0, a_1, a_2) = (0, 1, 2)$. In particular, for such a tuple, a_0 is interpreted as false while a_1 is interpreted as true. The unary relation $\texttt{Bool}$ will be forced by $\mathcal{Q}_1$ to contain at least a_0 and a_1.

Relations $\texttt{And}$, $\texttt{Or}$, and $\texttt{Neg}$ are intended to represent the respective logical functions. The first two attributes represent input values, and the last attribute represents the output.

Again, $\mathcal{Q}_1$ will guarantee that at least all triples of Boolean values that are consistent with the semantics of AND, OR, and NEG are present in these relations. Tuples in relation `Label` represent nodes together with their respective color (one can think of the representation of a node by ℓ-ary addresses over a ternary alphabet).
We define query $\mathcal{Q}_1$ as follows:

$$\begin{aligned} T() \leftarrow & \texttt{DomainValues}(w_0, w_1, w_2), \neg\texttt{DomainValues}(w_1, w_0, w_2), \\ & \neg\texttt{DomainValues}(w_2, w_1, w_0), \neg\texttt{DomainValues}(w_0, w_2, w_1), \\ & \texttt{Bool}(w_0), \texttt{Bool}(w_1), \texttt{Neg}(w_1, w_0), \texttt{Neg}(w_0, w_1), \\ & \texttt{And}(w_0, w_0, w_0), \texttt{And}(w_0, w_1, w_0), \texttt{And}(w_1, w_0, w_0), \texttt{And}(w_1, w_1, w_1), \\ & \texttt{Or}(w_0, w_0, w_0), \texttt{Or}(w_0, w_1, w_1), \texttt{Or}(w_1, w_0, w_1), \texttt{Or}(w_1, w_1, w_1). \end{aligned}$$

It is easy to see that $\mathcal{Q}_1$ enforces the conditions mentioned above.

In the following, we denote sequences $x_1, \ldots, x_\ell$ of ℓ variables by $\boldsymbol{x}$.
We define $\mathcal{Q}_2$ as the union of the queries $\mathcal{Q}_2^1$ and $\mathcal{Q}_2^2$, where subquery $\mathcal{Q}_2^1$ is defined as:

$$\begin{aligned} T() \leftarrow & \texttt{Bool}(x_1), \texttt{Bool}(x_2), \ldots, \texttt{Bool}(x_\ell), \texttt{DomainValues}(y_r, y_g, y_b), \\ & \neg\texttt{Label}(\boldsymbol{x}, y_r), \neg\texttt{Label}(\boldsymbol{x}, y_g), \neg\texttt{Label}(\boldsymbol{x}, y_b). \end{aligned}$$

Intuitively, $\mathcal{Q}_2^1$ can be satisfied in a database if for some node, represented by $\boldsymbol{x}$, there is no color.

Subquery $\mathcal{Q}_2^2$ deals with the correctness of a coloring and uses a set CIRCUIT of atoms that is intended to check whether for two nodes u and v, represented by $\boldsymbol{y}$ and $\boldsymbol{z}$, respectively, there is an edge between u and v.

To this end, CIRCUIT uses the variables $y_1, \ldots, y_\ell$, $z_1, \ldots, z_\ell$, representing the input and, at the same time, the 2ℓ input gates of C, and an additional variable u_i, for each gate of C, with the exception of the output gate. The output gate is represented by variable w_1. For each AND-gate represented by variable v_1 with incoming edges from gates represented by variables u_1 and u_2, CIRCUIT contains an atom $\texttt{And}(u_1, u_2, v_1)$. Likewise for OR- and NEG-gates.

Subquery $\mathcal{Q}_2^2$ is defined as:

$$T() \leftarrow \texttt{DomainValues}(w_0, w_1, w_2), \text{CIRCUIT}, \texttt{Label}(\boldsymbol{y}, u), \texttt{Label}(\boldsymbol{z}, u).$$

Intuitively, $\mathcal{Q}_2^2$ returns true when two nodes, witnessed to be adjacent by the circuit, have the same color.

Correctness of the reductions can be shown rather straightforwardly, as is done in the full version of this paper [14]. ◀

Next, we provide the above mentioned reduction.

▶ **Proposition 15.** $\text{CONTAINMENT}(\boldsymbol{BCQ}^\neg, \boldsymbol{UBCQ}^\neg) \leq_p \text{CONTAINMENT}(\boldsymbol{BCQ}^\neg, \boldsymbol{BCQ}^\neg)$.

Proof. Let $\mathcal{Q}_1$ be in $\mathbf{BCQ}^\neg$ and $\mathcal{Q}_2 = \bigcup_{i=1}^m \mathcal{Q}_2^i$ be in $\mathbf{UBCQ}^\neg$ over some database schema $\mathcal{D}$. Recall our assumption, that each disjunct is defined over a disjoint set of variables. Next, we construct CQs $\mathcal{Q}_1'$ and $\mathcal{Q}_2'$ such that $\mathcal{Q}_1' \subseteq \mathcal{Q}_2'$ if and only if, $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$.

We explain the intuition behind the reduction by means of an example. To this end, let $\mathcal{Q}_1$ be $H() \leftarrow A(x, y)$ and let $\mathcal{Q}_2$ be the $\mathcal{Q}_2^1 \cup \mathcal{Q}_2^2$, where $\mathcal{Q}_2^1$ is $H() \leftarrow A(u_1, v_1), B(u_1, v_1)$

and $\mathcal{Q}_2^2$ is $H() \leftarrow A(u_2, v_2), \neg B(u_2, v_2)$, both formulated over the schema $\mathcal{D} = \{A^{(2)}, B^{(2)}\}$. The query $\mathcal{Q}_2'$ takes the following form:

$$H() \leftarrow \texttt{Active}(x_0, x_1; \ell_1, \ell_2), \underbrace{\alpha(\ell_1, \mathcal{Q}_2^1)}_{\mathcal{Q}_{2,1}'}, \underbrace{\alpha(\ell_2, \mathcal{Q}_2^2)}_{\mathcal{Q}_{2,2}'},$$

where $\alpha(w, \mathcal{Q})$ denotes the modification of the body of $\mathcal{Q}$ by replacing every atom $\mathtt{R}(\boldsymbol{x})$ by $\mathtt{R}'(w, \boldsymbol{x})$. Both queries are defined over the schema $\mathcal{D}' = \{A'^{(3)}, B'^{(3)}, \texttt{Active}^{(4)}\}$. Notice that $\mathcal{Q}_2'$ contains a concatenation of the disjuncts of $\mathcal{Q}_2$. In addition, relations A and B are extended with a new first column with the purpose of labelling tuples. This labelling allows to encode two (or even more) instances over $\mathcal{D}$ by one instance over $\mathcal{D}'$. Specifically, $body_{\mathcal{Q}_1'}$ (not shown) is constructed in such a way that when there is a satisfying valuation for $\mathcal{Q}_1'$ there are two different data values, say 0 and 1. So, an instance I over $\mathcal{D}$ can be encoded as $I^0 = \{A'(0, a, b) \mid A(a, b) \in I\} \cup \{B'(0, a, b) \mid B(a, b) \in I\}$ or as $I^1 = \{A'(1, a, b) \mid A(a, b) \in I\} \cup \{B'(1, a, b) \mid B(a, b) \in I\}$. In addition, when there is a satisfying valuation for $\mathcal{Q}_1'$, there is an instance I_2 on which every disjunct of $\mathcal{Q}_2$ is true, and there is an instance I_1 on which $\mathcal{Q}_1$ is true. So, both $\mathcal{Q}_{2,1}'$ and $\mathcal{Q}_{2,2}'$ evaluate to true on I_2^0 when ℓ_1 and ℓ_2 are interpreted by label 0. However, for $\mathcal{Q}_1$ to be contained in $\mathcal{Q}_2$, we need that at least one of the disjuncts $Q_{2,1}'$ or $Q_{2,2}'$ evaluates to true over I_1^1, that is, when its labelling variable is interpreted as 1. Atom $\texttt{Active}(x_0, x_1; \ell_1, \ell_2)$ will ensure that x_0 and x_1 correspond with the values 0 and 1, and that at least one of the labelling variables ℓ_1 or ℓ_2 is equal to 1. In other words, `Active` chooses which disjunct to activate over I_1. So, at least one disjunct of $\mathcal{Q}_2$ evaluates to true on the instance I_1 on which $\mathcal{Q}_1$ is satisfied.

The reduction is explained in more detail in the full version of this paper [14]. ◀

Combining Propositions 14 and 15 we get the following corollary:

▶ **Corollary 16.** Containment($\boldsymbol{CQ}^\neg$, $\boldsymbol{CQ}^\neg$) *is* coNEXPTIME*-hard.*

The corresponding upper bounds hold also in the presence of inequalities and are shown by small model (i.e., counter-example) properties. To this end, we make use of a restricted monotonicity property of UCQ$^{\neg,\neq}$s which was already observed in Proposition 2.4 of [1]. For an instance I and a set D of data values we denote by $I_{|D}$ the restriction of I to facts that only use values from D.

▶ **Lemma 17** ([1])**.** *For $\mathcal{Q} \in \boldsymbol{UCQ}^{\neg,\neq}$, I an instance with a compatible schema, and D a set of data values, it holds that $\mathcal{Q}(I_{|D}) \subseteq \mathcal{Q}(I)$.*

Proof. Let $\boldsymbol{f} \in \mathcal{Q}(I_{|D})$ via a valuation V for a disjunct $\mathcal{Q}_i$ of $\mathcal{Q}$. Thus, $V(pos_{\mathcal{Q}_i}) \subseteq I_{|D} \subseteq I$. By definition, every variable x of $\mathcal{Q}_i$ occurs in a positive atom and therefore $V(x) \in D$. Thus, $V(neg_{\mathcal{Q}_i}) \cap I = V(neg_{\mathcal{Q}_i}) \cap I_{|D} = \emptyset$ and $\boldsymbol{f} \in \mathcal{Q}(I)$ as claimed. ◀

Now we can establish the following small model property for testing containment.

▶ **Lemma 18.** *Let $\mathcal{Q}_1, \mathcal{Q}_2 \in \boldsymbol{UCQ}^{\neg,\neq}$. If there is an instance I, where $\mathcal{Q}_1(I) \not\subseteq \mathcal{Q}_2(I)$, then there is also an instance $J \subseteq I$, where $\mathcal{Q}_1(J) \not\subseteq \mathcal{Q}_2(J)$, and $|adom(J)| \leq varmax(\mathcal{Q}_1)$.*

Proof. Let I be as in the lemma and let $\boldsymbol{f}$ be a fact with $\boldsymbol{f} \in \mathcal{Q}_1(I)$ and $\boldsymbol{f} \notin \mathcal{Q}_2(I)$. Let V be a valuation that derives $\boldsymbol{f}$ via some disjunct $\mathcal{Q}_1^i$ of $\mathcal{Q}_1$.

Let $D \stackrel{\text{def}}{=} adom(V(pos_{\mathcal{Q}_1^i}))$ and $J \stackrel{\text{def}}{=} I_{|D}$ the set of all facts in I using only values from $adom(V(pos_{\mathcal{Q}_i}))$. By definition, $|adom(J)| \leq varmax(\mathcal{Q}_1)$. Clearly, V is still a satisfying valuation for $\mathcal{Q}_1^i$ over J. However, by Lemma 17, $\boldsymbol{f} \notin \mathcal{Q}_2(J) = \mathcal{Q}_2(I_{|D})$. ◀

The upper bounds follow easily from Lemma 18.

▶ **Proposition 19.** *The following upper bounds hold:*
1. Containment($\boldsymbol{UCQ}^{\neg,\neq}, \boldsymbol{UCQ}^{\neg,\neq}$) *is in* coNEXPTIME.
2. *For every k, containment of $\boldsymbol{UCQ}^{\neg,\neq}$-queries over schemas with arity bound k is in Π_2^p.*

Proof. In both cases, we consider the complement of Containment($\mathbf{UCQ}^{\neg,\neq}, \mathbf{UCQ}^{\neg,\neq}$). Let $m \stackrel{\text{def}}{=} varmax(\mathcal{Q}_1)$.
1. A NEXPTIME algorithm, on input $\mathcal{Q}_1, \mathcal{Q}_2$, can simply guess an instance J with a domain of at most m elements and a fact $\boldsymbol{f}$, and verifies that $\boldsymbol{f} \in \mathcal{Q}_1(J)$ but $\boldsymbol{f} \notin \mathcal{Q}_2(J)$. For the latter tests, it can simply cycle, in exponential time, to all valuations over J for $\mathcal{Q}_1$ and $\mathcal{Q}_2$.
2. For a fixed arity bound, the minimal counter-example J is of size at most m^k. It can thus be guessed in polynomial time. That $\boldsymbol{f} \in \mathcal{Q}_1(J)$ can be verified non-deterministically. That $\boldsymbol{f} \notin \mathcal{Q}_2(J)$ can be verified by a universal computation in polynomial time. ◀

A claim of a Π_2^p upper bound for containment of CQs with negation can be found in [19]. It was not made clear there, that this claim assumes bounded arity of the schema. That the containment problem is Π_2^p-complete for schemas of bounded arity has been explicitly shown in [17]. Clearly, Proposition 19.2 follows directly and 19.1 is only a variation of it. From Proposition 19 and Corollary 16 the main result of this section immediately follows.

▶ **Theorem 20.** Containment($\boldsymbol{BCQ}^{\neg}, \boldsymbol{BCQ}^{\neg}$) *and* Containment($\boldsymbol{UCQ}^{\neg,\neq}, \boldsymbol{UCQ}^{\neg,\neq}$) *are* coNEXPTIME*-complete.*

Of course, the theorem also holds for all classes $\mathcal{C}$ of queries with $\mathbf{BCQ}^{\neg} \subseteq \mathcal{C} \subseteq \mathbf{UCQ}^{\neg,\neq}$.

6 Parallel-correctness: unions of conjunctive queries with negation

As mentioned in Section 4, for conjunctive queries without negation parallel-soundness always holds and thus parallel-correctness and parallel-completeness coincide, thanks to monotonicity. For queries with negation the situation is different. Distributed evaluation can be complete but not sound, or vice versa. For this reason, we have to distinguish all three problems separately: correctness, soundness, and completeness. However, the complexity is the same in all three cases.

Our results show a second, more crucial difference. Whereas parallel completeness for CQs without negation could be characterised in terms of valuations, that is, objects of polynomial size, our algorithms for CQs with negation involve counter-examples of exponential size (if the arity of schemas is not bounded) and the coNEXPTIME lower bound results indicate that this is unavoidable. We illustrate the observation that counter-examples might need an exponential number of tuples by the following example.

▶ **Example 21.** Let $\mathcal{Q}$ be the following conjunctive query with negation:

$$H() \leftarrow \texttt{Bool}(w_0, w_0), \texttt{Bool}(w_1, w_1), \texttt{Bool}(x_1, x_1), \ldots, \texttt{Bool}(x_n, x_n), \\ \neg\texttt{Bool}(w_0, w_1), \neg\texttt{Rel}(x_1, \ldots, x_n).$$

Let $\boldsymbol{P}$ be the policy defined over universe $U = \{0, 1\}$ and two-node network $\{\kappa_1, \kappa_2\}$, which distributes all facts except $\texttt{Rel}(0, \ldots, 0)$ to node κ_1 and only fact $\texttt{Rel}(0, \ldots, 0)$ to node κ_2.

Query $\mathcal{Q}$ is not parallel-sound under policy $\boldsymbol{P}$, but the smallest counter-example I is of exponential size as we argue next. Indeed, let $I \stackrel{\text{def}}{=} \{\texttt{Bool}(0,0), \texttt{Bool}(1,1)\} \cup \{\texttt{Rel}(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in \{0,1\}^n\}$. Furthermore, let valuation V map variables w_1 and w_0 to 1 and 0,

respectively, and map x_i to 0, for every $i \in \{1, \ldots, n\}$. Then, valuation V satisfies $\mathcal{Q}$ on instance $\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa_1) = I \setminus \{\texttt{Rel}(0, \ldots, 0)\}$ because neither $\texttt{Bool}(0,1)$ nor $\texttt{Rel}(0, \ldots, 0)$ is contained in the local instance. Furthermore, there is no satisfying valuation W for $\mathcal{Q}$ on the global instance I because W would have to map each x_i to either 0 or 1 implying that $W\big(\texttt{Rel}(x_1, \ldots, x_n)\big) \in I$.

However, there is no smaller instance: let I^* be some instance over universe U that has a locally satisfying valuation V. The combination of atoms $\texttt{Bool}(w_0, w_0), \texttt{Bool}(w_1, w_1)$, and $\neg\texttt{Bool}(w_0, w_1)$ in query $\mathcal{Q}$ then implies existence of both facts $\texttt{Bool}(0,0)$ and $\texttt{Bool}(1,1)$ because variables w_0 and w_1 cannot be mapped onto the same data value.

Assume that fact $\texttt{Rel}(a_1, \ldots, a_n)$, for some $(a_1, \ldots, a_n) \in \{0,1\}^n$ is missing from I^*. Then the valuation W that maps $w_0 \mapsto 0, w_1 \mapsto 1$ and $x_i \mapsto a_i$, for every $i \in \{1, \ldots, n\}$, satisfies $\mathcal{Q}$ also globally, on instance I^*, and can therefore be no example against parallel-soundness, which contradicts our choice of I^*. Thus, $\texttt{Rel}(a_1, \ldots, a_n) \in I^*$, for every $(a_1, \ldots, a_n) \in \{0,1\}^n$. We therefore have $I \subseteq I^*$ and, in particular, instance I^* contains at least as many facts as instance I. □

The results of this section are summarised in the following theorem:

▶ **Theorem 22.** *For every class $\mathcal{P} \in \{\mathcal{P}_{rule}\} \cup \mathfrak{P}_{npoly}$ of distribution policies, the following problems are* coNEXPTIME*-complete.*

- Parallel-Sound($\boldsymbol{UCQ}^{\neg}, \mathcal{P}$)
- Parallel-Complete($\boldsymbol{UCQ}^{\neg}, \mathcal{P}$)
- Parallel-Correct($\boldsymbol{UCQ}^{\neg}, \mathcal{P}$)

Theorem 22 follow from Propositions 23 and 25 below. It also holds for $\mathbf{UCQ}^{\neg,\neq}$. It is easy to show that, when restricted to schemas with some fixed (but sufficiently large, for hardness) arity bound, all these problems are Π_2^p-complete.

6.1 Upper bounds

In this section, we show the upper bounds of Theorem 22, summarised in the following proposition.

▶ **Proposition 23.** Parallel-Sound($\boldsymbol{UCQ}^{\neg,\neq}, \mathcal{P}$), Parallel-Complete($\boldsymbol{UCQ}^{\neg,\neq}, \mathcal{P}$), *and* Parallel-Correct($\boldsymbol{UCQ}^{\neg,\neq}, \mathcal{P}$) *are in* coNEXPTIME*, for every class $\mathcal{P} \in \mathfrak{P}$ of distribution policies. If the arity of schemas is bounded by some fixed number, these problems are in Π_2^p.*

Proof. As already indicated above, the proof relies on a bound on the size of a smallest counter-example. More specifically, we first show the following claim.

▶ **Claim 24.** Let $\mathcal{Q} \in \mathbf{UCQ}^{\neg,\neq}$ and let $\boldsymbol{P}$ be an arbitrary distribution policy. Then the following statements hold:

1. If $\mathcal{Q}$ is not parallel-complete under $\boldsymbol{P}$, then there is an instance J over a domain with at most $\mathit{varmax}(\mathcal{Q})$ elements such that $\mathcal{Q}$ is not parallel-complete on J under $\boldsymbol{P}$.
2. If $\mathcal{Q}$ is not parallel-sound under $\boldsymbol{P}$, then there is an instance J over a domain with at most $\mathit{varmax}(\mathcal{Q})$ elements such that $\mathcal{Q}$ is not parallel-sound on J under $\boldsymbol{P}$.

Towards (1) let us assume that $\mathcal{Q}$ is not parallel-complete on some instance I under $\boldsymbol{P}$. Let V be a valuation of a disjunct $\mathcal{Q}_i$ of $\mathcal{Q}$ that derives a fact $\boldsymbol{f}$ globally that is not derived on any node of the network. Let $D \stackrel{\text{def}}{=} \mathit{adom}(V(\mathit{pos}_{\mathcal{Q}_i}))$ and $J \stackrel{\text{def}}{=} I_{|D}$. Clearly, $|D| \leq \mathit{varmax}(\mathcal{Q})$ and V still derives $\boldsymbol{f}$ globally on instance J via $\mathcal{Q}_i$. On the other hand, for every node κ, $\mathcal{Q}\big(\mathit{loc\text{-}inst}_{\boldsymbol{P},J}(\kappa)\big) = \mathcal{Q}\big(\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa)_{|D}\big) \subseteq \mathcal{Q}\big(\mathit{loc\text{-}inst}_{\boldsymbol{P},I}(\kappa)\big)$, thanks to

Lemma 17. Therefore $\boldsymbol{f}$ is not derived on κ, and thus J witnesses the lack of parallel-completeness of $\mathcal{Q}$ under $\boldsymbol{P}$.

The proof of (2) is completely analogous. Given a counter-example I and a valuation V that derives a fact $\boldsymbol{f}$ on some node κ via $\mathcal{Q}_i$, for which $\boldsymbol{f}$ is not derived globally, we define $D \stackrel{\text{def}}{=} I_{|adom(V(pos_{\mathcal{Q}_i}))}$ and show that $J \stackrel{\text{def}}{=} I_{|D}$ is the desired counter-example.

An algorithm that tests the complement of parallel-completeness non-deterministically is described in the full version of this paper [14]. ◀

6.2 Lower bounds

The lower bounds stated in Theorem 22 follow from a polynomial time reduction from problem Containment($\mathbf{BCQ}^\neg, \mathbf{BCQ}^\neg$), for which we showed coNEXPTIME-hardness in Section 5.

▶ **Proposition 25.** Parallel-Complete($\boldsymbol{CQ}^\neg, \mathcal{P}_{rule}$), Parallel-Sound($\boldsymbol{CQ}^\neg, \mathcal{P}_{rule}$), *and* Parallel-Correct($\boldsymbol{CQ}^\neg, \mathcal{P}_{rule}$) *are* coNEXPTIME*-hard.*

Proof. Interestingly, all three results are shown by the *same* reduction from decision problem Containment($\mathbf{BCQ}^\neg, \mathbf{BCQ}^\neg$).

The basic idea for this reduction is very simple: it combines both queries $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbf{BCQ}^\neg$ of the given containment instance into a single query $\mathcal{Q} \in \mathbf{BCQ}^\neg$ and infers an appropriate distribution policy $\boldsymbol{P}$. To emulate separate derivation for both queries in the combined query, an activation mechanism is used that resembles the proof of Proposition 15. In this fashion, the two queries can be evaluated over different subsets of the considered instance by annotating both the facts in the instance as well as the atoms of the query.

We next describe the reduction in detail. Let thus $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbf{BCQ}^\neg$ be queries over some schema $\mathcal{D}$ and let $m \stackrel{\text{def}}{=} \max\{varmax(\mathcal{Q}_1), varmax(\mathcal{Q}_2)\}$. Without loss of generality, we assume the variable sets of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ to be disjoint. We will also assume in the following that both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are satisfiable. This is the case (for $\mathcal{Q}_1$) if and only if $pos_{\mathcal{Q}_1} \cap neg_{\mathcal{Q}_1} = \emptyset$ and can therefore be easily tested in polynomial time. If one of the test fails, some appropriate constant instance of Parallel-Complete($\mathbf{CQ}^\neg, \mathcal{P}_{\text{rule}}$) or one of the other problem variants, respectively, can be computed.

We define a (Boolean) query $\mathcal{Q} \in \mathbf{BCQ}^\neg$ and a policy $\boldsymbol{P} \in \mathcal{P}_{\text{rule}}$ over domain $\{1, \ldots, m\}$ that can be computed from $\mathcal{Q}_1$ and $\mathcal{Q}_2$ in polynomial time. The schema for $\mathcal{Q}$ is $\mathcal{D}' \stackrel{\text{def}}{=} \{R'^{(k+1)} \mid R^{(k)} \in \mathcal{D}\}$. That is, each relation name R of $\mathcal{D}$ occurs as R' in $\mathcal{D}'$ with an arity incremented by one. Additionally, $\mathcal{Q}$ uses relation names $\texttt{Type}$, $\texttt{Start}_1$, $\texttt{Start}_2$, and $\texttt{Stop}$, which we assume not to occur in schema $\mathcal{D}$. Besides the variables of $\mathcal{Q}_1$ and $\mathcal{Q}_2$, query $\mathcal{Q}$ uses variables ℓ_1, ℓ_2, t.

We use the function α, defined in the proof of Proposition 15, which adds its first parameter as first component to every tuple in its second parameter and translates relation names R into R'. In Proposition 15, the first parameter was always a variable and the second a set of atoms, but we use α also for a data value as first and a set of facts as second parameter in the obvious way. We write α_a^{-1} for the function mapping sets of facts over $\mathcal{D}'$ to sets of facts over $\mathcal{D}$, by selecting, from a set of facts, all facts with first parameter a, deleting this parameter and replacing each name R' by R. Finally, $\pi_a(I) \stackrel{\text{def}}{=} \alpha\big(a, \alpha_a^{-1}(I)\big)$ is the restriction of I to all facts with a in their first component.

The combined query $\mathcal{Q}$ has $head_{\mathcal{Q}} \stackrel{\text{def}}{=} H()$ and body

$$\begin{aligned} body_{\mathcal{Q}} \stackrel{\text{def}}{=}\ & \alpha(\ell_1, body_{\mathcal{Q}_1}) \cup \alpha(\ell_2, body_{\mathcal{Q}_2}) \\ \cup\ & \underbrace{\{\texttt{Type}(t), \texttt{Start}_1(\ell_1), \texttt{Start}_2(\ell_1), \texttt{Start}_2(\ell_2)\}}_{\mathcal{A}} \cup \underbrace{\{\neg\texttt{Stop}(\ell_1), \neg\texttt{Stop}(\ell_2)\}}_{\mathcal{A}^\neg}. \end{aligned}$$

Policy $\boldsymbol{P}$ is defined over universe $U \stackrel{\text{def}}{=} \{1, \ldots, m\}$, schema $\mathcal{D}' \cup \{\texttt{Type}, \texttt{Start}_1, \texttt{Start}_2, \texttt{Stop}\}$ and network $\mathcal{N} \stackrel{\text{def}}{=} \{\kappa_1, \ldots, \kappa_m, \sigma_1, \ldots, \sigma_m, \rho\}$. Facts are distributed as follows:

- Every node κ_i is responsible for the facts $\texttt{Type}(1), \texttt{Start}_1(i), \texttt{Start}_2(i), \texttt{Stop}(i)$, and all facts from $\mathit{facts}(\mathcal{D}', U)$.
- Every node σ_i is responsible for the facts $\texttt{Type}(2), \texttt{Start}_1(i), \texttt{Stop}(i)$, all $\texttt{Start}_2$-facts, and all facts from $\mathit{facts}(\mathcal{D}', U)$.
- Finally, node ρ is responsible for facts $\texttt{Type}(3), \ldots, \texttt{Type}(m)$, and *all* facts over other relation names.

It is easy to see that $\boldsymbol{P}$ can be expressed by a polynomial number of rules and that $\mathcal{Q}$ and $\boldsymbol{P}$ can be computed in polynomial time. In the full version of this paper [14], we show that the described function is indeed the desired reduction. ◀

7 Full conjunctive queries

In this section, we focus attention on full conjunctive queries, in an attempt to lower the complexity of testing parallel-correctness. Requiring queries to be full is a very natural restriction which is known to have practical benefits. For example, the Hypercube algorithm, which describes an optimal way to compute CQs in a setting very similar to ours, completely ignores projections when shuffling data, and only applies them when computing the query locally. The latter is possible because correctness for the full-variant of a query is in a sense more strict than correctness for the query itself.

Formally, a (union of) conjunctive queries is called *full* if all variables of the body also occur in the head. We denote by $\mathbf{FCQ}^{\neg,\neq}$ and $\mathbf{UFCQ}^{\neg,\neq}$ the class of full $\text{CQ}^{\neg,\neq}$ and full $\text{UCQ}^{\neg,\neq}$ queries, respectively, and likewise for other fragments.

The presentation is similar to that of Section 5 and 6. First, we establish the complexity of query containment. Then, we show that containment reduces to parallel-correctness (and variants). Finally, we obtain matching upper bounds.

The following theorem shows that unlike for general conjunctive queries the complexity of deciding containment for $\text{FCQ}^{\neg}$ and $\text{UFCQ}^{\neg}$ do not coincide.

▶ **Theorem 26.**
1. Containment($\boldsymbol{FCQ}^{\neg}, \boldsymbol{FCQ}^{\neg}$) *is in* P*;*
2. Containment($\boldsymbol{FCQ}^{\neg}, \boldsymbol{UFCQ}^{\neg}$) *is* coNP*-complete; and*
3. Containment($\boldsymbol{UFCQ}^{\neg}, \boldsymbol{UFCQ}^{\neg}$) *is* coNP*-complete.*

All these results also hold for queries with inequalities.

As one can reduce from Containment($\mathbf{FCQ}^{\neg}, \mathbf{UFCQ}^{\neg}$) to parallel-soundness, completeness, and correctness, we obtain the following hardness results:

▶ **Proposition 27.** Parallel-Sound($\boldsymbol{UFCQ}^{\neg}, \mathcal{P}$), Parallel-Complete($\boldsymbol{UFCQ}^{\neg}, \mathcal{P}$), *and* Parallel-Correct($\boldsymbol{UFCQ}^{\neg}, \mathcal{P}$) *are* coNP*-hard, for every* $\mathcal{P} \in \{\mathcal{P}_{rule}\} \cup \mathfrak{P}_{npoly}$.

The following theorem determines the complexity for the upper bounds:

▶ **Theorem 28.** *The following problems are* coNP*-complete:*
1. Parallel-Sound($\boldsymbol{UFCQ}^{\neg}, \mathcal{P}_{rule}$)*;*
2. Parallel-Complete($\boldsymbol{UFCQ}^{\neg}, \mathcal{P}_{rule}$)*;*
3. Parallel-Correct($\boldsymbol{UFCQ}^{\neg}, \mathcal{P}_{rule}$).

The result also holds for queries with inequalities.

8 Discussion

In this paper, we continued the study of parallel-correctness initiated by Ameloot et al. [4] as a framework for reasoning about one-round evaluation algorithms for conjunctive queries under arbitrary distribution policies. Specifically, we considered the case with union and negation. While parallel-correctness for unions of conjunctive queries can be tested by examining properties of single valuations, just like in the union-free case, the latter no longer holds true when negation is present. Consequently, we obtained that deciding parallel-correctness for unions of conjunctive queries remains in Π_2^p, while the analog problem in the presence of negation is hard for coNEXPTIME. Since conjunctive queries with negation are no longer monotone, we considered the related problems of parallel-completeness and parallel-soundness as well and obtained the same bounds. Interestingly, when negation is present, containment of conjunctive queries can be reduced to parallel-correctness (and its variants) allowing the transfer of lower bounds. We prove that containment for conjunctive queries with negation is hard for coNEXPTIME, which, to the best of our knowledge, is a novel result. In an attempt to lower complexity, we show that parallel-correctness for unions of full conjunctive queries with negation is coNP-complete.

There are quite a number of directions towards future work. While parallel-correctness for first-order logic is undecidable, it would be interesting to determine the exact frontier for decidability. As the considered problem is a static analysis problem that relates to the size of the queries and not to the size of the instances (at least in the setting of $\mathcal{P}_{\text{rule}}$), exponential lower bounds do not necessarily exclude practical application. It could still be interesting to identify settings that would make parallel-correctness tractable. Possibly independent of tractability considerations, such settings could incorporate bag semantics, integrity constraints, or specific classes (and representations) of distribution policies. We also plan to consider evaluation algorithms that use knowledge about the distribution policy to compute better query results, locally. Another direction for future work is to investigate transferability of parallel-correctness for conjunctive queries as defined in [4] in the presence of union and negation.

References

1 Foto N. Afrati, Stavros S. Cosmadakis, and Mihalis Yannakakis. On datalog vs. polynomial time. *J. Comput. Syst. Sci.*, 51(2):177–196, 1995. doi:10.1006/jcss.1995.1060.

2 Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. Parallel skyline queries. In *International Conference on Database Theory (ICDT 2012)*, pages 274–284, 2012. doi:10.1145/2274576.2274605.

3 Foto N. Afrati and Jeffrey D. Ullman. Optimizing joins in a map-reduce environment. In *Extending Database Technology (EDBT 2010)*, pages 99–110, 2010. doi:10.1145/1739041.1739056.

4 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and transferability for conjunctive queries. In *Principles of Database Systems (PODS 2015)*, pages 47–58. ACM, 2015.

5 Tom J. Ameloot, Bas Ketsman, Frank Neven, and Daniel Zinn. Weaker forms of monotonicity for declarative networking: a more fine-grained answer to the CALM-conjecture. In *Principles of Database Systems (PODS 2014)*, pages 64–75, 2014.

6 Tom J. Ameloot, Frank Neven, and Jan Van den Bussche. Relational transducers for declarative networking. *J. ACM*, 60(2):15, 2013. doi:10.1145/2450142.2450151.

7 Apache spark. URL: http://spark.apache.org.

8 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Principles of Database Systems (PODS 2013)*, pages 273–284, 2013.

9 Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In *Principles of Database Systems (PODS 2014)*, pages 212–223, 2014.

10 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Symposium on Theory of Computing (STOC 1979)*, pages 77–90, 1977.

11 Shumo Chu, Magdalena Balazinska, and Dan Suciu. From theory to practice: Efficient join query evaluation in a parallel database system. In *ACM SIGMOD Conference*, pages 63–78, 2015.

12 Carles Farré, Werner Nutt, Ernest Teniente, and Toni Urpí. Containment of conjunctive queries over databases with null values. In *International Conference on Database Theory (ICDT 2007)*, pages 389–403, 2007. doi:10.1007/11965893_27.

13 Sumit Ganguly, Abraham Silberschatz, and Shalom Tsur. Parallel bottom-up processing of datalog queries. *J. Log. Program.*, 14(1&2):101–126, 1992.

14 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and containment for conjunctive queries with union and negation. *CoRR*, abs/1512.06246, 2015. URL: http://arxiv.org/abs/1512.06246.

15 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *Principles of Database Systems (PODS 2011)*, pages 223–234, 2011.

16 Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *International Conference on Very Large Data Bases (VLDB 1993)*, pages 171–181, 1993.

17 Marie-Laure Mugnier, Geneviève Simonet, and Michaël Thomazo. On the complexity of entailment in existential conjunctive first-order logic with atomic negation. *Inf. Comput.*, 215:8–31, 2012.

18 Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986. doi:10.1016/S0019-9958(86)80009-2.

19 Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

20 Fang Wei and Georg Lausen. Containment of conjunctive queries with safe negation. In *International Conference on Database Theory (ICDT 2003)*, pages 343–357, 2003.

21 Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: SQL and rich analytics at scale. In *ACM SIGMOD Conference*, 2013.

22 Daniel Zinn, Todd J. Green, and Bertram Ludäscher. Win-move is cordination-free (sometimes). In *International Conference on Database Theory (ICDT 2012)*, pages 99–113, 2012.

A Formal Study of Collaborative Access Control in Distributed Datalog

Serge Abiteboul[1], Pierre Bourhis[2], and Victor Vianu[3]

1 **INRIA Saclay, Palaiseau, France; and**
ENS Cachan, Cachan, France
serge.abiteboul@inria.fr

2 **CNRS, France; and**
Centre de Recherche en Informatique, Signal et Automatique (CRIStAL) – UMR 9189, Lille, France
pierre.bourhis@univ-lille1.fr

3 **University of California San Diego, La Jolla, USA; and**
INRIA Saclay, Palaiseau, France
vianu@cs.ucsd.edu

Abstract

We formalize and study a declaratively specified collaborative access control mechanism for data dissemination in a distributed environment. Data dissemination is specified using distributed datalog. Access control is also defined by datalog-style rules, at the relation level for extensional relations, and at the tuple level for intensional ones, based on the derivation of tuples. The model also includes a mechanism for "declassifying" data, that allows circumventing overly restrictive access control. We consider the complexity of determining whether a peer is allowed to access a given fact, and address the problem of achieving the goal of disseminating certain information under some access control policy. We also investigate the problem of information leakage, which occurs when a peer is able to infer facts to which the peer is not allowed access by the policy. Finally, we consider access control extended to facts equipped with provenance information, motivated by the many applications where such information is required. We provide semantics for access control with provenance, and establish the complexity of determining whether a peer may access a given fact together with its provenance. This work is motivated by the access control of the Webdamlog system, whose core features it formalizes.

1998 ACM Subject Classification H.2.0 [Database Management] General – Security, integrity and protection, H.2.1 [Database Management] Logical Design – Data models, H.2.3 [Database Management] Languages – Data description languages, data manipulation languages

Keywords and phrases Distributed datalog, access control, provenance

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.10

1 Introduction

The personal *data* and favorite *applications* of Web users are typically distributed across many heterogeneous devices and systems. In [19], a novel *collaborative access control mechanism* for a distributed setting is introduced in the context of the language Webdamlog, a datalog-style language designed for autonomous peers [3, 2]. The experimental results of [19] indicate that the proposed mechanism is practically feasible, and deserves in-depth investigation. In the present paper, we provide for the first time formal grounding for the mechanism of [19] and answer basic questions about the semantics, expressiveness, and computational cost of such a mechanism. In the formal development, we build upon *distributed datalog* [16, 20],

which abstracts the core of Webdamlog, while ignoring certain features, such as updates and delegation.

In this investigation, as in Webdamlog, access control is *collaborative* in the following sense. The system provides the *means* to specify and infer access rights on disseminated information, thus *enabling* peers to collectively enforce access control. The system is agnostic as to how peers are motivated or coerced into conforming to the access control policy. This can be achieved in various ways, from economic incentives to legal means (see, e.g., [28]), possibly relying on techniques such as encryption or watermarking (see, e.g., [5]). We do not address these aspects here.

The access control of [19] that we formalize and study here works as follows. First, each peer specifies which other peers may access each of its extensional relations using *access-control-list rules*. This provides in a standard manner an initial coarse-grained (relation-at-a-time) access control, enforced locally by each peer. Next, facts can be derived among peers using *application rules*. Access control is extended to such facts based on their provenance: to see a propagated fact, a peer must have access to the extensional relations used by the various peers in producing the fact. This enables controlling access to data *disseminated* throughout the entire network, at a fine-grain (i.e., tuple) level. This capability is a main distinguishing feature of Webdamlog's access control model. The access control also includes a *hide* mechanism that allows circumventing overly restrictive access control on some disseminated facts, thus achieving a controlled form of "declassification" for selected peers.

Access control in distributed datalog raises a variety of novel semantic, expressiveness and complexity issues. How complex is it to check whether a peer has the right to access a propagated fact? What are the appropriate complexity measures in this distributed setting? Does the access control mechanism prevent leakage of unauthorized information? What does it mean to extend access control to facts equipped with their *provenance*? Is there an additional cost? These are some of the fundamental questions we study, described in more detail next.

While the experimental results of [19] suggest that the computational cost of the proposed mechanism is modest, we show formally that its complexity is reasonable. Specifically, we prove that the data complexity of determining whether a peer can access a given fact is PTIME-complete (with and without *hide*).

We next consider the problem of information leakage, which occurs when a peer is able to infer some facts to which the peer is not allowed access by the policy. We show that, while undecidable in general, information leakage can be tested for certain restricted classes of policies and is guaranteed not to occur for more restricted classes.

One of the challenges of access control is the intrinsic tension between access restrictions and desired exchange of information. We consider the issue of achieving the goal of disseminating certain information under some access control policy. The goal is specified as a distributed datalog program. We show that it is undecidable whether a goal can be achieved without declassification (i.e., without *hide*). We study the issue of finding a policy without *hide* that achieves a maximum subset of the specified goal. While any goal can be achieved by extensive use of *hide*, we show, more interestingly, how this can be done with *minimal* declassification.

In many applications, it is important for inferred facts to come with *provenance* information, i.e., with traces of their derivation. We demonstrate that adding such a requirement has surprising negative effects on the complexity. For this, we introduce an intermediate measure between data and combined complexity, called *locally-bounded* combined complexity that

allows making finer distinctions than the classical measures in our context. The intuition is that the peers are seen as part of the data and not of the schema, which is more in the spirit of a Web setting. We show that the locally bounded complexity of query answering increases from PTIME-complete to PSPACE-complete when it is required that the query answer carries *provenance* information.

The organization is as follows. Section 2 recalls the distributed datalog language [3]. In Section 3, we formalize the core aspects of the access control mechanism of [19], establish the complexity of answering queries under access control. Information leakage is studied in Section 4. The issue of achieving some dissemination goal under a particular access control policy is the topic of Section 5. Access control in the presence of provenance is investigated in Section 6. Finally, we discuss related work and conclude.

2 Distributed Datalog

In this preliminary section, we formally define a variant of distributed datalog, which captures the core of Webdamlog [3].

The language. We assume infinite disjoint sets Ext of extensional relation symbols, Int of intensional relation symbols, $\mathcal{P}$ of *peers* (e.g. p, q), $\mathcal{D}_p$ of *pure data values* (e.g., a, b), and $\mathcal{V}$ of *variables* (e.g., x, y, X, Y). For relations, we use symbols such as R, S, T. The set $\mathcal{D}$ of *constants* is $\mathcal{P} \cup \mathcal{D}_p \cup Ext \cup Int$. A *schema* is a mapping σ whose domain $dom(\sigma)$ is a finite subset of $\mathcal{P}$, that associates to each p a finite set $\sigma(p)$ of relation symbols in $Int \cup Ext$, with associated arities. Let σ be a schema, $p \in dom(\sigma)$. A relation R in $\sigma(p)$ is denoted by $R@p$, and its arity by $arity(R@p)$. We denote $ext(p) = \sigma(p) \cap Ext$, $int(p) = \sigma(p) \cap Int$, $ext(\sigma) = \cup_{p \in dom(\sigma)} ext(p)$, and $int(\sigma) = \cup_{p \in dom(\sigma)} int(p)$. An *instance* I over σ is a mapping associating to each relation schema $R@p$ a finite relation over $\mathcal{D}$ of the same arity. For a tuple $\bar{a}$ in $I(R@p)$, the expression $R@p(\bar{a})$ is called a (p-)*fact* in $R@p$. An *extensional* instance is one that is empty on $int(\sigma)$. Observe that $R@p$ and $R@q$, for distinct p, q, are distinct relations with no a priori semantic connection, and possibly different arities. Note also that an expression $R@p(a_1, ..., a_k)$ for $R, p, a_1, ..., a_k$ in $\mathcal{D}$ is a *fact for* a schema σ if: p is a peer in $dom(\sigma)$, R is a relation schema in $\sigma(p)$, and $arity(R@p) = k$. Note that relations may contain pure data values, peers, as well as relation symbols. Finally, (U)CQ denotes (unions) of conjunctive queries (see [6]).

▶ **Definition 1** (distributed datalog). A *d-datalog* program P over schema σ is a finite set of rules of the form

- $Z_0@z(\bar{x}_0) :- R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$ where
- $p \in dom(\sigma)$, $k \geq 0$, and for every $i \geq 1$, R_i is in $\sigma(p)$ and $\bar{x}_i$ is a vector of variables and constants in $\mathcal{D}$ of the proper arity;
- $z \in dom(\sigma) \cup \mathcal{V}$, $Z_0 \in Int \cup \mathcal{V}$; and
- each variable occurring in the head appears in $\bar{x}_i$ for some $i \geq 1$.

Note that the relation or peer names in the head may be variables. Note also that all the relations in the body of a rule come from the same peer. Although we define a global d-datalog program, one should think of each peer p as having its separate program consisting of all the rules whose bodies use relations at p.

▶ **Example 2.** Consider the rules:

- $Album@Alice(x) :- Album@Bob(x)$
- $Album@z(x) :- Album@Bob(x), Friend@Bob(z)$
- $Z@z(x) :- Album@Bob(x), FriendPhotos@Bob(Z, z)$

Bob uses the first rule to publish his photos in Alice's album, and the second to publish his photos in all of his friends' albums (peer variable z). In the last rule, different names can be used for the relations where the friends keep their photos (variable Z for a relation name).

A d-datalog program defines the meaning of intensional relations from given extensional relations. The semantics is in the spirit of the datalog semantics. More precisely:

▶ **Definition 3** (Semantics)**.** Let P be a d-datalog program over some schema σ. The *immediate consequence operator* Γ_P on instances over σ is defined as follows. Let I be an instance over σ.

Consider a rule $Z_0@z(\bar{x}_0) :- R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$ of P. An *instantiation* of the rule in I is a mapping ν from its variables to the active domain (the set of values occurring in P, I, or $dom(\sigma)$), extended with the identity on constants, such that:

- for each $i \geq 1$, $R_i@p(\nu(\bar{x}_i)) \in I$; and
- $\nu(Z_0)@\nu(z)(\nu(\bar{x}_0))$ is a fact for schema σ.

$\Gamma_P(I)$ is obtained by adding to I all facts $\nu(Z_0)@\nu(z)(\nu(\bar{x}_0))$ where ν is an instantiation in I of some rule $Z_0@z(\bar{x}_0) :- R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$ of P. Note that Γ_P is monotonic. The *semantics* of P for an extensional instance I, denoted $P(I)$, is the mapping associating to each extensional instance I the *projection* on the intensional relations of P of the *least fixpoint* of Γ_P containing I.

Observe that a rule may "attempt" to derive an improper fact, for which $\nu(z)$ is not in $dom(\sigma)$, or $\nu(Z_0)$ is not a relation in $\sigma(\nu(z))$, or the arity is incorrect. In such cases, the fact is simply *not* derived.

▶ Remark. Consider a rule with variable peer or relation name. Suppose for instance that both are variables. A *head-instantiation* ν of that rule for a schema σ is a mapping over Z_0, z such that $\nu(z)$ is a peer of σ, $\nu(Z_0)$ an intensional relation of $\sigma(\nu(z))$, and $arity(\nu(Z_0)) = |\bar{x}_0|$. One can define similarly the notion of head-instantiation for a rule with only a variable peer or only a variable relation name. It is easy to see that the program obtained by replacing each rule by all its head-instantiations has the same semantics as the original. So if the set of peers is fixed (known in advance), one can assume that, for each rule, the name of the relation and the peer in the head are constants.

3 The access control model

In this section, we formalize the core aspects of the access control mechanism of [19]. The focus here is on the READ privilege; we will ignore the GRANT privilege (allowing a peer to define permissions on another peer's relations) and the WRITE privilege (allowing a peer to push data to another peer's relations), see [19]. We also provide in this section basic expressiveness and complexity results on access control.

The extensional relations at a given peer are owned by the peer. The peer can give READ privilege on these extensional relations to other peers. This is specified at each peer p using an intensional relation $acl@p$ (for *access control list*) of arity 2. A fact $acl@p(R, q)$ states that peer q is allowed to read the extensional relation $R@p$.

In the following, we assume that for each peer p, $acl \in int(p)$ and $arity(acl@p) = 2$. For instance, a rule "$acl@p(R, z) :- Likes@p(z)$" can be used in a program to grant access to relation $R@p$ to all the peers z that are in relation $Likes@p$.

A d-datalog *program* P with access control (denoted d-datalog$_{ac}$) over some schema σ is a finite set of d-datalog rules $Z_0@z(\bar{x}_0) :- R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$, where $R_1, ..., R_k$ are not acl and the rules are of one of the following two kinds:

- **Application rule:** Z_0 is not acl; and
- **Access control rule:** The rule head is $acl@p(Z, z)$ for some terms Z, z.

Given a program P, the set of application rules forms the *application program* of P, denoted P_{app}, and the set of access control rules forms the *(access control) policy* of P, denoted P_{pol}. Facts of the form $acl@p(R, q)$ are called *access control facts*, and the others are called *application facts*. It should be noted that no such distinction is made in Webdamlog. We distinguish here between access control and application rules to be able to formally compare access control policies.

The meaning of an access control policy P_{pol} for a given extensional instance I is clear in the absence of intensional relations: use the access rules to compute at each peer the set of peers allowed to read its extensional relations. This yields relation-at-a-time, coarse-grained access control to the extensional relations. For intensional relations, we use tuple-level fine-grained access control. Intuitively, an intensional fact can be read by a peer p if it can be derived by some application of a rule from tuples that p is already allowed to access. Then, for a d-datalog$_{ac}$ program P, P_{app} and P_{pol} may interact recursively: the derivation of an intensional fact may yield some new permission for an extensional relation, which, in turn, may enable the derivation of a new intensional fact, and so on. The fine-grained access control at the tuple level is illustrated in an example.

▶ **Example 4.** Consider the program P:

$$
\begin{array}{lll}
P_{pol} & acl@Bob(Album, z) & :\!- friends@Bob(z); \\
 & acl@Bob(Tagged, z) & :\!- friends@Bob(z); \\
P_{app} & Album@z(x) & :\!- Album@Bob(x), Tagged@Bob(x, z);
\end{array}
$$

The access control rules allow Bob's friends access to his *Album* and *Tagged* relations. The application rule transfers to a given person the photos in which he/she is tagged. Consider a photo α with tagging *Sue*, assuming she is a friend of Bob. Then the picture α belongs (intensionally) to Sue's album. A friend of Bob who will ask to see Sue's album will see the photo α.

With standard access control, peers are only be able to control access to their local data. With the proposed mechanism, they further control the *dissemination* of their data. In other words, they can control what *other* peers should do with their data. This is achieved by propagating, together with data, permissions via application rules, based on *provenance* information about derived facts. A tuple derived by some instantiation of an application rule is accessible by a peer if that peer has access to each tuple in the body of the rule.

The semantics. To define the semantics of programs, we associate with each peer p in $dom(\sigma)$ and each relation $R@p$, $R \neq acl$, a relation $\widehat{R}@p$ of arity $arity(R) + 1$. Intuitively, $\widehat{R}@p(\bar{x}, q)$ says that peer q is allowed access to the fact $R@p(\bar{x})$. The semantics is defined using a d-datalog program. We describe next the construction of that program.

▶ **Definition 5** ($\widehat{P}$ construction)**.** The semantics of a d-datalog$_{ac}$ program P over some schema σ for an extensional instance I over σ is defined using a d-datalog program $\widehat{P}$ (without access control) defined as follows. Its schema consists of: (i) the extensional and intensional relations of σ; and (ii) intensional relations $\{\widehat{R}@p \mid R@p \in \sigma(p), R \neq acl\}$.

The rules of $\widehat{P}$ are as follows: for a tuple $\bar{x}$ of distinct variables,

1. $\widehat{R}@p(\bar{x}, p) :\!- R@p(\bar{x})$ for each peer p in σ and each $R \in ext(p)$ (each peer can read its own extensional relations);

2. $\widehat{R}@p(\bar{x}, z) :- acl@p(R, z), R@p(\bar{x})$ for each peer p in σ and each $R \in ext(p)$ (each peer z entitled to read $R@p$ can read all of its tuples);
3. for each rule $acl@p(Z, z) :- R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$ in P_{pol},
   a rule $acl@p(Z, z) :- \widehat{R}_1@p(\bar{x}_1, p), \cdots, \widehat{R}_k@p(\bar{x}_k, p)$;
4. for each rule $Z_0@z(\bar{x}_0) :- R_1@p(\bar{x}_1), \cdots, R_k@p(\bar{x}_k)$ in P_{app} and for each intensional relation $R_0 \neq acl$ occurring in σ, a rule[1]
   $\widehat{R}_0@z(\bar{x}_0, y) :- Z_0 = R_0, \widehat{R}_1@p(\bar{x}_1, y), \cdots, \widehat{R}_k@p(\bar{x}_k, y), \widehat{R}_1@p(\bar{x}_1, z), \cdots, \widehat{R}_k@p(\bar{x}_k, z)$
5. A rule $R@p(\bar{x}) :- \widehat{R}@p(\bar{x}, p)$ for each $p \in dom(\sigma)$ and $R \in int(p)$ ($\widehat{R}@p$ defines the local facts visible at p).

The fourth item requires that both z (the next reader) and y (potential future readers) may access the facts in the body of the rule, in order be allowed to see the fact derived by the rule. The third item is the analog for *acl*. Note that (3.) is simpler than (4.) because the relation *acl* is only defined locally.

Clearly, the size of $\widehat{P}$ is linear in P and the image of σ. Moreover, it is independent of the data, i.e. $dom(\sigma)$ and I. Using $\widehat{P}$, we define two semantics for P: state semantics, and visibility semantics.

State semantics. State semantics provides for each peer the *local* intensional facts inferred by taking into account the combined effect of the access control rules and the application rules. More precisely, the state semantics of a d-datalog$_{ac}$ program P over schema σ is a mapping $[P]$ associating to each extensional instance I over σ the set of facts

$$[P](I) = \{R@p(\bar{a}) \in \widehat{P}(I) \mid p \in dom(\sigma), R \in int(p)\}.$$

One can easily verify by induction that $[P](I) \subseteq P(I)$. (Recall that $P(I)$ is the access-control-free semantics). The inclusion may be strict because the derivation of a fact at a peer p may be blocked because p does not have access to some data.

Visibility semantics. This semantics captures more broadly the facts at *all* peers that a given peer is allowed to see. Indeed, in addition to their local state provided by $[P]$, peers also have permission to see facts residing at *other* peers. The facts that they are allowed to see are specified by the relations $\widehat{R}@q(-, p)$ defined by the program $\widehat{P}$. We say that such a fact is *visible* by a peer p. For each p, we denote by $[P]_p^{\mathcal{V}}$ the mapping associating to each instance I over $ext(\sigma)$ the set of facts $\{R@q(\bar{u}) \mid \widehat{R}@q(\bar{u}, p) \in \widehat{P}(I)\}$. We refer to $[P]_p^{\mathcal{V}}$ as the *visibility semantics* for peer p. Clearly, for each p, $[P]_p^{\mathcal{V}}(I)$ and $[P](I)$ agree on $int(p)$.

Intuitively, if a fact $R@q(\bar{a})$ is visible by p, then p can access it by querying the relation $R@q$. More precisely, let P' be the program obtained by adding to P a rule $temp@p(\bar{u}) :- R@q(\bar{u})$ for some new relation $temp@p$ and vector $\bar{u}$ of distinct variables. Then $temp@p(\bar{a}) \in [P'](I)$ iff $R@q(\bar{a}) \in [P]_p^{\mathcal{V}}(I)$, i.e. $R@q(\bar{a})$ is visible by p. Thus, visibility semantics can be reduced to state semantics by the addition of such rules.

In addition to state and visibility semantics, we consider in Section 4 the facts that a peer may *infer* from the visible ones, possibly circumventing the access control policy. We will refer to this as *implicit visibility*.

[1] Strictly speaking, equalities $Z = R_0$ are not allowed in d-datalog, but these can be easily simulated by substituting the variable by the constant everywhere in the rule.

Hiding access restrictions. The above access control mechanism may be too constraining in some situations. We next consider means of relaxing it. To do so, we introduce a *hide* annotation that can be attached to atoms in rule bodies, e.g., [hide $R@q(\bar{x})$]. Intuitively, such an annotation lifts access restrictions on $R@q(\bar{x})$ by "hiding its provenance".

We illustrate this feature with an example.

▶ **Example 6.** Consider the two rules:

- $Album@z(x)$:– $Album@Bob(x), friend@Bob(z)$
- $Album@z(x)$:– $Album@Bob(x),$ [hide $friend@Bob(z)$]

The first rule is used by Bob to publish his photos in all of his friends albums. Suppose Sue is a friend. Will the photos in $Album@Bob$ be transferred to $Album@Sue$? Yes, but only if Sue has read privileges on both $Album@Bob$ and $friends@Bob$. However, it may be the case that Bob wishes to keep his list of friends private, but still let his friends see his album pictures. He can do this by "hiding" the access restrictions on $friends@Bob$ as in the second rule. Intuitively, Bob is in effect reducing the protection level of the *friend* relation, in some sense "declassifying" it.

In the example, Bob declassifies *his own* extensional relation. As we will see, "hide" also allows a peer to declassify information received from *other* peers, thus overriding their access control restrictions. In the actual Webdamlog system [19], doing so requires the peer to have GRANT privilege on that piece of information. As previously mentioned, for simplicity we do not consider explicitly the GRANT mechanism here.

For further illustration, we show how the hide mechanism can be used to simulate accessing a relation with binding patterns [24].

▶ **Example 7.** Suppose that peer p wishes to export an extensional binary relation R with binding pattern bf. The intuition is that one cannot obtain the entire relation, but if one provides bindings for the first column, peer p will provide the corresponding values in the second column. This is done as follows:

- $Seed@p(x)$:– $S@q(x)$
- $Q@q(x,y)$:– $Seed@p(x),$ [hide $R@p(x,y)$]

Suppose the access control policy is such that p has read privilege on $S@q$, but q has no read privilege on $R@p$. Observe that $Seed@p$ is a copy of $S@q$, and $Q@q$ is the join of $Seed@p$ and $R@p$. Peer q cannot see $R@p$. But if q provides some values for the first column of $R@p$ (in relation $S@q$), then q will obtain in $Q@q$ the corresponding values for the second column of $R@p$.

Programs with *hide* are defined as follows.

▶ **Definition 8.** A *d-datalog$_{ac}$ program with hide* (denoted h-d-datalog$_{ac}$) over some schema σ consists of: (i) a d-datalog$_{ac}$ program $P = P_{app} \cup P_{pol}$; and (ii) a function h (called the *hide* function) whose domain h is the set P_{app} of rules[2], such that for each rule r, $h(r)$ is a strict subset of the atoms in the body of r. The pair (P_{pol}, h) forms the *policy* of the program.

As in Example 6, the function h is represented using annotations. More precisely, in each rule, the atoms in $h(r)$ are annotated with the keyword *hide*. For instance, the rule r that is A :– $B_1, \ldots B_5$ with $h(r) = \{B_2, B_4\}$ is denoted: A :– $B_1,$ [hide B_2], $B_3,$ [hide B_4], B_5.

We next consider how hide annotations modify the semantics of access control. The semantics for h-d-datalog$_{ac}$ programs is obtained by replacing item (4) of Definition 5 with:

[2] Because of the way we define access control rules, *hide* annotations would have no effect on them.

4'. for each application rule $Z_0@z(\bar{x}_0) :- R_1@p(\bar{x}_1), \cdots, R_k@p(\bar{x}_k)$ of P_{app}, for each intensional relation $R_0 \neq acl$ occurring in σ, and some new variable y, the rule $\widehat{R}_0@z(\bar{x}_0, y) :- Z_0 = R_0, \widehat{R}_1@p(\bar{x}_1, y_1), \cdots \widehat{R}_k@p(\bar{x}_k, y_k), \widehat{R}_1@p(\bar{x}_1, q_1), \cdots \widehat{R}_k@p(\bar{x}_k, q_k)$ where for each i, if $R_i@p(\bar{x}_i)$ is not hidden in the rule, $y_i = y$ and $q_i = z$; and if it is hidden, $y_i = q_i = p$.

Note that this imposes that both y (a potential future reader) and z (the site that will host the fact) can read the facts in the body of the rule *that are not annotated by hide*, in order for the reader to be allowed to see the fact derived by the rule. For a h-d-datalog$_{ac}$ program P, we denote by $[P]$ the state semantics of P as defined by the above program.

The next result, namely Proposition 10, shows that the use of hide extends the expressive power of d-datalog$_{ac}$ relative to state semantics. (One can obtain a similar result for visibility semantics.) This is illustrated by the following example.

▶ **Example 9.** Consider a peer p that has a binary extensional relation $R@p$. Suppose we wish to specify that peer q sees from $R@p$ exactly the tuples of the form $(x, 0)$, and no other peer sees anything from $R@p$. As a first attempt, one might use an intensional relation R_{export} and the rule: $R_{export}@q(x, 0)$:- $R@p(x, 0)$.

However, either $acl@p(R, q)$ holds, so $R@p$ is entirely visible to q; or not, and $R_{export}@q$ is empty. Considering *hide*, assume the existence of some extensional fact $ok_q@p()$ that only q can read. Then there is a solution: $R_{export}@q(x, 0) :- ok_q@p(), [\text{hide } R@p(x, 0)]$.

▶ **Proposition 10.** *There is a h-d-datalog$_{ac}$ program P over schema σ for which there is no d-datalog$_{ac}$ program $\bar{P}$ such that, for every extensional instance I over σ, $[P](I) = [\bar{P}](I)$.*

Thus, the *hide* construct strictly increases the expressivity of the language. In fact, we will show in Section 5 that h-d-datalog$_{ac}$ is in some sense expressively complete.

The complexity of access control. We consider throughout the paper the complexity of various problems related to access control. Typically, three kinds of complexity are considered in databases: data, query, and combined complexity. In d-datalog$_{ac}$, the distinction between data and schema/program is less clear. For instance, the set of peers affects both the schema and the data. If there are many peers, the global program may be large, even if each peer has a small program. To capture this situation, we consider a measure assuming that the size of the program *at each peer* is bounded. This gives rise to a novel notion of complexity that we call *locally-bounded combined complexity*. More precisely, for a decision problem whose input is an extensional instance I and a d-datalog$_{ac}$ program P over some schema σ:

- The *combined complexity* is computed as a function of $|I|$, $|P|$, and σ.
- The *data complexity* is computed as a function of $|I|$ only (σ and P are fixed).
- The *locally-bounded combined complexity* is computed as a function of $|I|$ and $|dom(\sigma)|$, assuming some fixed bound on the size of the program at each peer (so $|P|$ is linear in the number of peers).

We begin by establishing the complexity of checking the visibility of a fact.

▶ **Theorem 11.** *Let σ be a schema, I an extensional instance, and P a h-d-datalog$_{ac}$ program over σ. Determining whether a fact is in $[P]^{\mathcal{V}}_p(I)$ for some peer p has* PTIME*-complete data and locally-bounded combined complexity, and* EXPTIME*-complete combined complexity,*

While the data and the locally-bounded combined complexities are the same in this case, we will see later that the two differ in other settings, allowing to draw finer distinctions than the classical notions.

Static analysis of policies. To conclude this section, we briefly discuss the issue of *comparing* policies relative to a given application program, based on the visible facts they allow. This leads to the notion of a policy being *more relaxed than* another. By reduction from containment of datalog programs, one can show that this is undecidable for given policies and application program. As for datalog containment, one can consider restrictions for which the policy comparison can be performed, e.g., "frontier-guarded" rules [8]. As an alternative to comparing policies, one can consider applying syntactic transformations to a given policy in order to relax or tighten it. For example, augmenting the hide function of a program, or adding rules to P_{pol}, always results in a more relaxed policy. Due to space limitations, we do not further consider these issues here.

4 Implicit visibility

The purpose of access control is to analyse the ability of peers to see unauthorized information. As discussed in Section 3, a peer can access information by examining its own state or by querying relations of other peers. But can a peer infer more information beyond what is allowed according to the policy? We capture this using the notion of *implicit visibility* (i-visibility) that we formalize next. For this, we use the auxiliary notion of "visibility instance". For a program P over σ and a peer p, we say that an instance I_p over σ is a *visibility instance* of p if there is some instance J over $ext(\sigma)$ for which $I_p = [P]_p^{\mathcal{V}}(J)$. Now we define:

▶ **Definition 12.** Let P be a d-datalog$_{ac}$ program over some schema σ, p a peer and I_p a visibility instance for p. A fact $R@q(\bar{u})$ (for some q, R) is *i(mplicitly)-visible at p* given I_p, if for each instance J over $ext(\sigma)$ such that $[P]_p^{\mathcal{V}}(J) = I_p$, $R@q(\bar{u}) \in J \cup [P](J)$.

It turns out that facts beyond $[P]_p^{\mathcal{V}}(J)$ may be i-visible at peer p. To see how such information "leakage" can occur, suppose that we have a rule $acl@q(R, p) :- Q@q(p)$, where $Q@q$ is an extensional relation. If peer p sees some fact in $R@q$, it can infer that it has access to $R@q$, so that $Q@q(p)$ holds, although the policy may not allow p to see $Q@q$. This may in turn provide additional information on other relations. Before exploring this formally, we introduce some restrictions of policies.

▶ **Definition 13.** Let σ be a schema and $P = P_{pol} \cup P_{app}$ a d-datalog program.

- The policy of P is *static* iff for each rule of P_{pol}, its body is empty;
- The policy of P is *simple* iff for each rule of P_{pol}, the atoms in its body are extensional;
- The policy of P is *local* for P_{app} iff for each peer p and rule of P_{pol} at p, the atoms in its body are either extensional, or intensional but not depending on non-local relations.

We can show that with static policy, no leakage can occur.

▶ **Proposition 14.** *Let P be a d-datalog$_{ac}$ program over σ with static policy. For each peer p and instance I over $ext(\sigma)$, the set of i-visible facts at p is precisely $[P]_p^{\mathcal{V}}(I)$.*

In contrast to the above, when P_{pol} contains arbitrary rules, i-visibility provides additional information, and is in fact undecidable.

▶ **Theorem 15.** *It is undecidable, given a d-datalog$_{ac}$ program P over σ, a visibility instance I_p for p, and a fact $R@q(\bar{u})$, whether $R@q(\bar{u})$ is i-visible at p given I_p. Moreover, undecidability holds even for programs with local access policies.*

The above undecidability result uses the fact that the *acl* relations are defined by datalog programs. We next show that i-visibility becomes decidable if recursion is disallowed in the definition of *acl* relations. The problem can be reduced to computing certain answers to datalog queries using exact UCQ views, which is known to be in co-NP [4]. However, using the fact that the views we use are particular UCQs, we can show that the complexity goes down to PTIME.

▶ **Theorem 16.** *The i-visibility problem for d-datalog$_{ac}$ programs with simple policies is decidable in* PTIME *(data complexity).*

The i-visibility problem with hide. We now turn to the problem of i-visibility for d-datalog$_{ac}$ programs with hide. The notions of visibility and i-visibility are adapted to this setting in the natural way. We first illustrate the fact that hide can lead to non-trivial i-visibility of facts, even when the acl policy is static.

▶ **Example 17.** Consider the following h-d-datalog$_{ac}$ program P where P_{pol} consists of the rule $acl@q(Q,p)$:- and P_{app} of the rules:

- $R_1@p(X)$:– $Q@q(), [\text{ hide } R@q(X,Y)]$;
- $R_2@p(Y)$:– $Q@q(), [\text{ hide } R@q(X,Y)]$.

Consider the p-visibility instance $\{R_1@q(a), R_2@q(b)\}$. Note that p does not have access to $R@q$. However, it is clear that $R@q(a,b)$ is i-visible at p.

The following result shows that i-visibility is undecidable for h-d-datalog$_{ac}$ programs even for static policies (when, by Proposition 14, no leakage occurs in the absence of *hide*). The proof is by reduction from finding certain answers to identity queries using exact datalog views, known to be undecidable [4].

▶ **Theorem 18.** *It is undecidable, given a h-d-datalog$_{ac}$ program P over σ with static policy, in which* hide *is applied only to extensional relations, a peer p, a p-visibility instance I_p, and an extensional fact $R@q(\bar{a})$, whether $R@q(\bar{a})$ is i-visible at p given I_p.*

Testing information leakage. The previous result concerned i-visibility for a given instance. We finally consider the problem of testing whether a d-datalog$_{ac}$ program has information leakage beyond that provided by the access control policy for *some* instance (the static analysis analog).

▶ **Definition 19.** A d-datalog$_{ac}$ program P *leaks information* at p if for some p-visibility instance I_p there exists some fact $R@q(\bar{a}) \notin I_p$ that is i-visible at p given I_p.

We show that one cannot generally decide whether a program leaks information. However, one can do so for programs with *simple* policies. The undecidability is proved using a reduction from datalog program containment. The 2EXPTIME algorithm for simple policies is by reduction to an exponential set of inclusions of datalog programs into UCQs.

▶ **Theorem 20.**

1. *It is undecidable, given a d-datalog$_{ac}$ program P and a peer p, whether P leaks information at p.*
2. *The problem is* 2EXPTIME*-complete if P has a simple acl policy.*

5 Achieving dissemination goals

We next consider the problem of achieving a specific data dissemination goal among peers, when a particular access control policy is imposed. The goal is specified by a d-datalog program. Clearly, a given goal may violate the policy, so it may be impossible to achieve it. We study the problem of determining whether achieving a goal is possible, and if not, how one might maximize what *can* be achieved. We then consider the issue of relaxing the access control policy in order to achieve the goal, using the *hide* mechanism. Not surprisingly, it is always possible to achieve a goal using *hide*. More interestingly, we will show how to do so while minimizing its use. But first, we consider what can be done without *hide*.

Strict adherence to the policy. Consider a policy P_{pol} and a goal d-datalog program P. We wish to know whether there is a d-datalog program P_{app} such that (i) P_{app} uses the relations of P and possibly additional intensional relations, and (ii) for each extensional instance I, $[(P_{pol} \cup P_{app})](I)$ and $P(I)$ agree on the intensional relations of P. In this case, we say that P_{app} *simulates* P under policy P_{pol}. We will see that it is generally impossible to find such a P_{app} without *hide*, and present restrictions on the policies that make it possible. When such a simulation does not exist, we will attempt to find a program that is as close as possible to the goal.

The next example illustrates how a policy may prevent achieving a goal even in the simplest setting. The example is more complicated than needed because we will also use it to illustrate finding a "maximum" simulation.

▶ **Example 21.** Consider the following policy and goal program:

$$\begin{array}{llll} P_{pol} & acl@p(R_1, r) :- & ; & P \quad R@q(x) :- R_1@p(x); \\ & acl@p(R_2, r) :- & ; & \quad\; R@q(x) :- R_2@p(x); \\ & acl@p(R_1, q) :- & ; & \quad\; R@r(x) :- R@q(x) \end{array}$$

The d-datalog P does not simulate P under P_{pol} because q is not allowed to see the relation $R_2@p$ and therefore the relation $R@q$ does not hold tuples from $R_2@p$ under the policy P_{pol}. In such cases, we can try to find a program that is, in some sense, maximally achieves the goal. This is a nontrivial issue. In this example, a maximum application program is:

$$P_{app}: \quad R@q(x) :- R_1@p(x); \mid R@r(x) :- R@q(x); \mid R@r(x) :- R_2@p(x).$$

Note that $[(P_{pol} \cup P_{app})] \subseteq P$ but $[(P_{pol} \cup P)] \subset [(P_{pol} \cup P_{app})]$ (as mappings).

The first result states that one cannot decide whether a program can be simulated under a particular policy.

▶ **Theorem 22.** *It is undecidable, given a policy P_{pol} and a goal d-datalog program P, whether there exists a d-datalog program P_{app} without* hide *such that P_{app} simulates P under P_{pol}. This holds even if P_{pol} is static.*

If such a simulation is not possible, can we find a "maximum simulation"? Let P be a d-datalog program over some schema σ and P_{pol} a policy program over σ. A d-datalog program P_{app} without *hide* is a *maximum simulation* of P under P_{pol} iff

1. $[(P_{pol} \cup P_{app})] \subseteq P$, and
2. for each P'_{app} such that $[(P_{pol} \cup P'_{app})] \subseteq P$, $[(P_{pol} \cup P'_{app})] \subseteq [(P_{pol} \cup P_{app})]$.

The question of whether a maximum simulation always exists remains open. Moreover, there does not exist an algorithm building a maximum simulation, *if such exists*.

▶ **Theorem 23.** *There is no algorithm that computes, given a d-datalog program P and a policy P_{pol}, a maximum simulation without* hide *P_{app} of P under P_{pol}, whenever such a maximum simulation exists. This holds even for local policies.*

While it is not known whether a maximum simulation always exists, we present informally a plausible candidate for a maximum simulation of P under P_{pol} and explore its potential. The program, denoted by $\text{MAC}(P_{pol}, P)$, is based on a simple idea: each peer collects all the extensional tuples that peer is allowed to see under P_{pol}, and then simulates P locally.

▶ **Definition 24.** Let P be a d-datalog program over some schema σ and P_{pol} a policy over the relations in σ. The program $P_{app} = \text{MAC}(P_{pol}, P)$ is constructed as follows:

1. For all peers p, q, $p \neq q$ and each (extensional or intensional) relation $R@q$, P_{app} has an intensional relation $R\_q@p$ of the same arity as $R@q$. These relations allow p to perform a simulation of P with the data that p has access to.
2. For all peers p, q, $p \neq q$, and each extensional relation $R@q$, P_{app} has rules copying $R@q$ into $R\_q@p$, if $acl@q(R, p)$ holds.
3. Finally, for each peer p, P_{app} has rules that simulate P locally with the data that p has access to.

Observe how $\text{MAC}(P_{pol}, P)$ interacts with P_{pol}. During the computation, some peer p may use rules in P_{pol} to derive a new fact $acl@p(R, q)$. This results in copying $R@p$ into $R\_p@q$ which may lead to the derivations of more facts at p.

Note the connection between $\text{MAC}(P_{pol}, P)$ and P itself. By definition, $[(P_{pol} \cup P)] \subseteq [(P_{pol} \cup \text{MAC}(P, P_{pol}))]$. However, the inclusion may be strict. For instance, P may try to transfer a fact from p to q via a peer r that is not allowed to see this fact whereas it is possible to send this fact directly (with a different rule) without violating access rights.

It turns out, surprisingly, that $\text{MAC}(P_{pol}, P)$ is not always a maximum simulation of P under P_{pol}, and it is in fact undecidable whether $\text{MAC}(P_{pol}, P)$ is a maximum simulation for some given (P_{pol} and P, even for local policies. However, $\text{MAC}(P_{pol}, P)$ is a maximum simulation if P_{pol} is static.

▶ **Theorem 25.** *Let P_{pol} be a local policy and P a d-datalog goal program over σ.* (i) *It is undecidable whether the program $\text{MAC}(P, P_{pol})$ is a maximum simulation of P under P_{pol}.* (ii) *If P_{pol} is static, then $\text{MAC}(P, P_{pol})$ is a maximum simulation of P under P_{pol}.*

Besides ensuring the existence of a maximal simulation, a simple policy is of interest for another reason: it guarantees that, if there exists some application program simulating P under P_{pol}, then P itself simulates P under that policy (details omitted).

Declassifying information. Let us now consider the issue of achieving a goal at the cost of declassifying information, in other words using the *hide* construct. There is an immediate solution that would consist in modifying every rule of the goal program P by hiding the entire body. The goal would be satisfied, but in a brutal way: each derived fact would be visible to all peers.

It is possible to realize the goal in a much more controlled way as illustrated by Example 9. In that exemple, special relations of the form $ok_q@p$ are used to limit as much as possible the visibility of data. The example suggests the following mild technical assumptions: (†) for all distinct peers $p, q \in dom(\sigma)$, (1) σ contains a 0-ary extensional relation $ok_q@p$, and (2) extensional instances of σ are assumed to contain the fact $ok_q@p()$.

We next show that (†) is sufficient to guarantee that the *hide* construct allows achieving any goal program by declassifying no more information than necessary.

▶ **Theorem 26.** *Let σ satisfy* (†.1)*. For each policy P_{pol} and a d-datalog goal program P over σ, there exists an application P_{app} with* hide *over the same σ such that, for each extensional instance I satisfying* (†.2)*, P_{app} simulates P under P_{pol}; and on input I, a fact $R@p(u)$ is visible at $q \neq p$ for $(P_{pol} \cup P_{app})$ iff it is visible at q for $(P_{pol} \cup P)$.*

6 Accessing provenance

We considered so far the inference of individual facts using d-datalog$_{ac}$ rules, subject to an access control policy. In many applications, it is essential for inferred facts to be accompanied by *provenance* information. In this section, we extend our approach to access control to cover provenance. We adopt a simple model of provenance of a fact, consisting of derivation trees tracing the application of the rules at different peers that participated in the inference of the fact. To simplify the presentation, we ignore *hide*. The definition of provenance can be easily adapted to the presence of *hide* (a *hide* annotation in a rule results in truncating the corresponding portion of the proof tree) and the complexity results continue to hold.

Consider a d-datalog$_{ac}$ program P over schema σ. Let I be an extensional instance over σ, and $R@p(\bar{a})$ a fact in $P_{app}(I)$. A *provenance tree* for $R@p(\bar{a})$ is a derivation tree for $R@p(\bar{a})$ using P_{app} and I. Intuitively, we are interested in passing provenance information from peer to peer, so that a peer p not only knows that some fact $R@p(u)$ holds, but can also know how $R@p(u)$ has been derived.

▶ **Example 27.** Consider a schema σ with peers $\{p_0, p_1, p_2, p_3, p_4\}$, 0-ary extensional relations (propositions), $R@p_0$, $R@p_1$, and 0-ary intensional relations $S@p_2$, $S@p_3$, $S@p_4$. Let $I = \{R@p_0, R@p_1\}$. Consider the following application program:

$$P_{app} \quad S@p_2 :- R@p_0; \mid S@p_2 :- R@p_1; \mid S@p_3 :- S@p_2; \mid S@p_4 :- S@p_3.$$

Note that $S@p_4 \in P_{app}(I)$ and has two provenance trees (linear in this case):

$$S@p_4 \leftarrow S@p_3 \leftarrow S@p_2 \leftarrow R@p_1 \qquad\qquad S@p_4 \leftarrow S@p_3 \leftarrow S@p_2 \leftarrow R@p_0$$

Suppose we have the following access control rules in addition to P_{app}:

$$P_{pol}: \quad acl@p_0(R, p_2) :- ; \mid acl@p_0(R, p_4) :- ; \mid acl@p_1(R, p_3) :- ; \mid acl@p_1(R, p_4) :- .$$

Consider again the two provenance trees of $S@p_4 \in P_{app}(I)$. Neither satisfies the access control policy defined by P_{pol}. Indeed, the first tree violates the policy because p_2 does not have access to $R@p_1$. The second also violates the policy, because p_3 does not have access to $R@p_0$. If we add the access control rule: $acl@p_0(R, p_3)$:– then the second provenance tree satisfies the access control policy.

Note the difference between visibility of a fact A by a peer p and visibility of its *provenance*. In order for A to be visible by p, it suffices for each fact involved in its derivation to be visible by the corresponding intermediate peer, based on its own access permissions, independently derived. In other words, peers may justify their permissions by derivations independent of each other and of the actual derivation of A. Visibility of provenance imposes a stronger condition, as it requires each intermediate peer to have access to the *entire history* of the partial derivation of p. As seen in the example, a fact A may itself be visible by p but not have any provenance tree visible by p. More formally we have:

▶ **Definition 28** (Provenance access control)**.** Let P be a d-datalog$_{ac}$ program over some schema σ and I an extensional instance over σ. A fact F *has visible provenance* if there exists a provenance tree T of F such that: For each internal node $R@p(\bar{a})$ in T and extensional fact $E@q(\bar{c})$ occurring in the subtree rooted at $R@p(\bar{a})$, we have that $acl@q(E, p) \in [P](I)$. For given P and I, $[P]^{prov}(I)$ denotes the set of facts that have visible provenance.

It is clear that visible provenance implies visibility. More precisely, one can show that for each P, σ, and each extensional instance I, $[P]^{prov}(I) \subseteq [P](I)$, but Example 27 shows the converse does not hold. We next show that, although the definition of provenance visibility is proof-theoretic, one can simulate it using a d-datalog program. However, unlike the program $\widehat{P}$ constructed earlier, the program simulating provenance visibility is exponential in the number of peers.

▶ **Proposition 29.** *Let P be a d-datalog$_{ac}$ program over some schema σ. There exists a d-datalog program (without access control) P^{prov} of size exponential in* $\mathrm{dom}(\sigma)$ *(and polynomial in σ and P if* $\mathrm{dom}(\sigma)$ *is fixed) with the same extensional relations as σ, such that for each extensional instance I, $[P]^{prov}(I)$ and $P^{prov}(I)$ agree on the intensional relations of σ.*

The program P^{prov} (in the proof of the previous result) uses constants to denote sets of peers. An alternative would consist in using an extension of d-datalog with nesting, in the style of extensions of datalog with nesting [6]. (Such a nested datalog is used in the implementation in [19].)

The d-datalog program P^{prov} is exponential in the set $dom(\sigma)$ of peers. Is it possible to avoid the exponential blowup? The following complexity result implies a negative answer (subject to usual assumptions). Consider the problem of deciding, given an extensional instance I and a program P, whether a fact is in $[P]^{prov}(I)$. Recall from Theorem 11 that the complexity of checking visibility of a fact has EXPTIME-complete combined complexity, and PTIME-complete data and locally-bounded combined complexity. Now we have:

▶ **Theorem 30.** *Let σ be a schema, I an extensional instance, and P a d-datalog$_{ac}$ program over σ. Determining whether a fact is in $[P]^{prov}(I)$ has* EXPTIME*-complete combined complexity,* PTIME*-complete data complexity and* PSPACE*-complete locally-bounded combined complexity.*

Theorems 11 and 30 show that provenance visibility has the same combined and data complexity as the standard semantics, but different locally-bounded combined complexity. As a corollary, the exponential blowup in Proposition 29 cannot be avoided (unless PTIME = PSPACE). This highlights the usefulness of this complexity measure in making finer distinctions than the classical ones.

7 Related work

Database security and access control have been studied in depth (e.g., see [10]) since the earliest works on System R [26] and Ingres [27].

Controlling access to intensional facts in deductive languages is related to managing virtual views in SQL, which is handled differently among various database systems. When an authorized user accesses a view, it is usually evaluated with the privileges of the defining user ("definer's rights"). Some systems (e.g. mySQL) allow the creator of a view to specify that later access to the view will be with respect to the privileges of the invoker of the view ("invoker's rights"). This is similar in spirit to our approach.

The access control model we have described is fine-grained, unlike the SQL standard. Lefevre et al [18] propose a fine-grained access control model for implementing personal privacy policies in a relational database. They use query modification to enforce their policies, as we do, but their policy model and implementation are oriented towards a centralized database system. A commercial example of fine-grained access control is Oracle's Virtual Private Database (VPD), which supports access control at the level of tuples or cells. VPD

allows an administrator to associate an external function with a relation and automatically modifies queries to restrict access by tuple or cell. Alternative semantics for fine-grained access control have been investigated thoroughly [18, 25, 29]. Rizvi et al. [25] distinguish between Truman and Non-Truman models (the expression is motivated by the movie *The Truman Show* where the hero is unaware that he lives in an artificial environment). Query answers in our system follow the Truman paradigm: queries are not rejected because of lack of privilege but the user's privileges limit the answers that are returned.

Fine-grained access control is also studied in [13], where predicate-based specification of authorization is supported. The inference of sensitive data from exposed data (that we study here under the name of i-visibility) is related to a notion studied in [30].

Our model of access control shares some features with the model of reflective database access control (RDBAC) in which access policies can be specified in terms of data contained in any part of the database. Olson et al. [21] formalize RDBAC using a version of datalog with updates [11] but their model does not include distribution, delegation, or the use of provenance. In Cassandra [17], access rights are specified using a language based on datalog with constraints. The language supports complex specifications based on "user roles". On the other hand, fine-grained access control is not considered.

The use of provenance as a basis for access control was first noted in the context of provenance semirings [15, 7]. A security semiring can contain tuple-level security annotations and define the rules by which they are propagated to query results. Another example of provenance-based access control is the work of Park et al. [23] in which access decisions are based on a transactional form of provenance.

The emergence of social networks and other Web 2.0 applications has led to new forms of access control. In online social networks, the distinguishing feature is that access control policy is expressed in terms of network relationships amongst members [12, 14], and this is one of the motivations of the model we presented. However, the model is intended to support the diverse requirements of access control in a variety of distributed applications.

The Webdamlog language was first described in [3] as a version of distributed datalog in which peers exchange not only facts, but also rules. Expressiveness and semantic issues were formally investigated, but access control was not considered. As already mentioned, we build here on the Webdamlog access control mechanism of [19]. Its main novelty is the specification of the access rights on an inferred tuple based on the access rights on the tuples used to derive it. The full access control mechanism of [19] is richer than the one described here, notably using also GRANT and WRITE privileges. They present an open-source implementation (with Bud [9] inside), and an experimental evaluation showing that the computational cost of access control is modest. In the Webdam project context, cryptographic techniques for enforcing access control in a distributed manner (and detecting security violations) have been considered in [5]. The techniques proposed there can be combined with those presented here.

Security in distributed systems has primarily focused on issues of remote authentication, authorization, and protection of data and distributed trust; such issues are outside the scope of our present work [1, 22].

8 Conclusion

We presented a first formal study of provenance-based access control in distributed datalog inspired by the collaborative access control mechanism of [19]. The results highlight the subtle interplay between declarative distributed computation, coarse-grained and fine-grained access control. Starting from coarse-grained access control on local extensional relations,

distributed datalog computation yields fine-grained access control on derived facts based on their provenance. We also considered access control on tuples equipped with explicit provenance. We briefly studied the problem of information leakage, occurring when peers can infer unauthorized information from authorized data. We established the complexity of access control, as well as of various analysis tasks, such as detecting information leakage, comparing access policies, or the ability to achieve specified goals under a given policy. A challenging aspect of the framework is the fluid boundary of schema, data, and program, that has an impact on both semantics and complexity. For example, this led us to define a new complexity measure, locally-bounded combined complexity, that can make more subtle distinctions than classical data and query complexity.

In this first investigation, we have ignored some important aspects of the Webdamlog system presented in [19]. In Webdamlog, "nonlocal rules" allow dynamic deployment of rules from one peer to another. Most of the results presented here extend to non-local rules. We also ignored here the GRANT and WRITE privileges of Webdamlog. These raise new subtle issues, notably when access control updates are considered. Finally, delegation in Webdamlog allows peers to assign tasks to other peers. The access control of delegation is supported in Webdamlog by a mechanism called "sandboxing" that also raises interesting issues. These are left for future research.

Acknowledgement. The authors gratefully acknowledge the support of the French ANR Aggreg project ANR-14-CE25-0017 (Pierre Bourhis) and of the U.S. National Science Foundation under award IIS-1422375 (Victor Vianu).

References

1 Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. In *ACM Trans. Program. Lang. Syst.*, pages 706–734, 1993.

2 Serge Abiteboul, Emilien Antoine, Gerome Miklau, Julia Stoyanovich, and Jules Testard. [Demo] rule-based application development using WebdamLog. In *SIGMOD*, 2013.

3 Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Emilien Antoine. A rule-based language for Web data management. In *PODS*, 2011.

4 Serge Abiteboul and Oliver M Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263. ACM, 1998.

5 Serge Abiteboul, Alban Galland, and Neoklis Polyzotis. A model for web information management with access control. In *WebDB Workshop*, 2011.

6 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

7 Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.

8 Vince Bárány, Balder ten Cate, and Martin Otto. Queries with guarded negation. *PVLDB*, 5(11):1328–1339, 2012.

9 Berkeley Orders Of Magnitude Project. Bloom programming language. URL: `http://www.bloom-lang.net/`.

10 Elisa Bertino and Ravi Sandhu. Database security-concepts, approaches, and challenges. *Dependable and Secure Computing, IEEE Transactions on*, 2(1):2–19, 2005.

11 Anthony Bonner. Transaction datalog: A compositional language for transaction programming. In *DBPL*. Springer, 1997.

12 Barbara Carminati, Elena Ferrari, Raymond Heatherly, Murat Kantarcioglu, and Bhavani Thuraisingham. A semantic web based framework for social network access control. In *SACMAT*, pages 177–186, 2009. doi:10.1145/1542207.1542237.

13 Surajit Chaudhuri, Tanmoy Dutta, and S Sudarshan. Fine grained authorization through predicated grants. In *ICDE*, pages 1174–1183. IEEE, 2007.

14 Elena Ferrari. *Access Control in Data Management Systems*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

15 Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

16 Guy Hulin. Parallel processing of recursive queries in distributed architectures. In *VLDB*, pages 87–96, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

17 Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, 2010.

18 Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases. In *VLDB*, pages 108–119. VLDB Endowment, 2004.

19 Vera Zaychik Moffitt, Julia Stoyanovich, Serge Abiteboul, and Gerome Miklau. Collaborative access control in WebdamLog. In *SIGMOD*, 2015.

20 Wolfgang Nejdl, Stefano Ceri, and Gio Wiederhold. Evaluating recursive queries in distributed databases. *Knowledge and Data Engineering, IEEE Transactions on*, 5(1):104–121, 1993.

21 Lars E. Olson, Carl A. Gunter, and P. Madhusudan. A formal framework for reflective database access control policies. In *CCS'08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 289–298, New York, NY, USA, 2008. ACM. doi:10.1145/1455770.1455808.

22 M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

23 Jaehong Park, Dang Nguyen, and R. Sandhu. A provenance-based access control model. In *International Conference on Privacy, Security and Trust*, pages 137–144, 2012. doi:10.1109/PST.2012.6297930.

24 Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D Ullman. Answering queries using templates with binding patterns. In *PODS*, pages 105–112. ACM, 1995.

25 Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, pages 551–562, New York, NY, USA, 2004. ACM Press. doi:10.1145/1007568.1007631.

26 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 *SIGMOD Conference*, pages 23–34, 1979.

27 Michael Stonebraker, Gerald Held, Eugene Wong, and Peter Kreps. The design and implementation of INGRES. *ACM Trans. Database Syst.*, 1(3):189–222, September 1976. doi:10.1145/320473.320476.

28 Prasang Upadhyaya, Magdalena Balazinska, and Dan Suciu. Automatic enforcement of data use policies with datalawyer. In *SIGMOD*, pages 213–225, 2015.

29 Qihua Wang, Ting Yu, Ninghui Li, Jorge Lobo, Elisa Bertino, Keith Irwin, and Ji-Won Byun. On the correctness criteria of fine-grained access control in relational databases. In *VLDB*, pages 555–566, 2007.

30 Hong Zhu, Jie Shi, Yuanzhen Wang, and Yucai Feng. Controlling information leakage of fine-grained access model in dbmss. In *WAIM*, pages 583–590. IEEE, 2008.

It's All a Matter of Degree: Using Degree Information to Optimize Multiway Joins

Manas R. Joglekar[1] **and Christopher M. Ré**[2]

1 **Department of Computer Science, Stanford University, Stanford, CA, USA**
`manasrj@stanford.edu`
2 **Department of Computer Science, Stanford University, Stanford, CA, USA**
`chrismre@stanford.edu`

Abstract

We optimize multiway equijoins on relational tables using degree information. We give a new bound that uses degree information to more tightly bound the maximum output size of a query. On real data, our bound on the number of triangles in a social network can be up to 95 times tighter than existing worst case bounds. We show that using only a constant amount of degree information, we are able to obtain join algorithms with a running time that has a smaller exponent than existing algorithms – *for any database instance*. We also show that this degree information can be obtained in nearly linear time, which yields asymptotically faster algorithms in the serial setting and lower communication algorithms in the MapReduce setting.

In the serial setting, the data complexity of join processing can be expressed as a function $O(\text{IN}^x + \text{OUT})$ in terms of input size IN and output size OUT in which x depends on the query. An upper bound for x is given by fractional hypertreewidth. We are interested in situations in which we can get algorithms for which x is strictly smaller than the fractional hypertreewidth. We say that a join can be processed in subquadratic time if $x < 2$. Building on the AYZ algorithm for processing cycle joins in quadratic time, for a restricted class of joins which we call 1-series-parallel graphs, we obtain a complete decision procedure for identifying subquadratic solvability (subject to the 3-SUM problem requiring quadratic time). Our 3-SUM based quadratic lower bound is tight, making it the only known tight bound for joins that does not require any assumption about the matrix multiplication exponent ω. We also give a MapReduce algorithm that meets our improved communication bound and handles essentially optimal parallelism.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Joins, Degree, MapReduce

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.11

1 Introduction

We study query evaluation for natural join queries. Traditional database systems process joins in a pairwise fashion (two tables at a time), but recently a new breed of multiway join algorithms have been developed that satisfy stronger runtime guarantees. In the sequential setting, worst-case-optimal sequential algorithms such as NPRR [16,17] or LFTJ [18] process the join in runtime that is upper bounded by the largest possible output size, a stronger guarantee than what traditional optimizers provide. In MapReduce settings[1], the Shares algorithm [2,13] processes multiway joins with optimal communication complexity on skew

[1] A description of Background material including MapReduce, as well as proofs of all our results, can be found in the full version of the paper [12]

Table 1 Triangle bounds on various social networks.

Network	MO **Bound**	AGM **Bound**	$\frac{\text{AGM}}{\text{MO}}$
Twitter	$225M$	$3764M$	17
Epinions	$33M$	$362M$	11
LiveJournal	$6128M$	$573062M$	95

free data. However, traditional database systems have developed sophisticated techniques to improve query performance. One popular technique used by commercial database systems is to collect "statistics": auxiliary information about data, such as relation sizes, histograms, and counts of distinct different attribute values. Using this information helps the system better estimate the size of a join's output and the runtimes of different query plans, and make better choices of plans. Motivated by the use of statistics in query processing, we consider how statistics can improve the new breed of multiway join algorithms in sequential and parallel settings.

We consider the first natural choice for such statistics about the data: the degree. The degree of a value in a table is the number of rows in which that value occurs in that table. We describe a simple preprocessing technique to facilitate the use of degree information, and demonstrate its value through three applications: i) An improved output size bound ii) An improved sequential join algorithm iii) An improved MapReduce join algorithm. Each of these applications has an improved exponent relative to their corresponding state-of-the-art versions [5, 8, 16, 18].

Our key technique is what we call *degree-uniformization*. Assume for the moment that we know the degree of each value in each relation, we then partition each relation by degree of each of its attributes. In particular, we assign each degree to a bucket using a parameter L: we create one bucket for degrees in $[1, L)$, one for degrees in $[L, L^2)$, and so on. We then place each tuple in every relation into a partition based on the degree buckets for each of its attribute values. The join problem then naturally splits into smaller join problems; each smaller problem consisting of a join using one partition from each relation. Let IN denote the input size, if we set $L = \text{IN}^c$ for some constant c, say $\frac{1}{4}$, the number of smaller joins we process will be exponential in the number of relations – but constant with respect to the data size IN. Intuitively, the benefit of joining partitions separately is that each partition will have more information about the input and will have reduced skew. We show that by setting L appropriately this scheme allows us to get tighter AGM-like bounds.

Now we consider a concrete example. Suppose we have a d-regular graph with N edges; the number of triangles in the graph is bounded by $\min(Nd, \frac{N^2}{d})$ by our degree-based bound and by $N^{3/2}$ by the AGM bound. In the worst case, $d = \sqrt{N}$ and our bound matches the AGM bound. But for other degrees, we do much better; better even than simply "summing" the AGM bounds over each combination of partitions. Table 1 compares our bound (MO) with the AGM bound for the triangle join on social networks from the SNAP datasets [14]. 'M' in the table stands for millions. The last column shows the ratio of the AGM bound to our bound; our bound is tighter by a factor of $11x$ to $95x$. We could not compare the bounds on the Facebook network, but if the number of friends per user is ≤ 5000, our bound is at least $450x$ tighter than the AGM bound.

We further use degree uniformization as a tool to develop algorithms that satisfy stronger runtime and communication guarantees. Degree uniformization allows us to get runtimes with a better exponent than existing algorithms, while requiring only linear time preprocessing on the data. We demonstrate our idea in both the serial and parallel (MapReduce) setting, and we now describe each in turn.

Serial Join Algorithms: We use our degree-uniformization to derive new cases in which one can obtain subquadratic algorithms for join processing. More precisely, let IN denote the size of the input, and OUT denote the size of the output. Then the runtime of an algorithm on a query Q can be written as $O(\text{IN}^x + \text{OUT})$ for some x. Note that $x \geq 1$ for all algorithms and queries in this model as we must read the input to answer the query. If the query is α-acyclic, Yannakakis' algorithm [19] achieves $x = 1$. If the query has fractional hypertree width (fhw), a recent generalization of tree width [10], equal to 2, then we can achieve $x = 2$ using a combination of algorithms like NPRR and LFTJ with Yannakakis' algorithm. In this work, we focus on cases for which $x < 2$, which we call *subquadratic algorithms*. Subquadratic algorithms are interesting creatures in their own right, but they may provide tools to attack the common case in join processing in which OUT is smaller than IN.

Our work builds on the classical AYZ algorithm [4], which derives subquadratic algorithms for cycles using degree information. This is a better result than the one achieved by the fhw result since the fhw value of length ≥ 4 cycles is already $= 2$. This result is specific to cycles, raising the question: *"Which joins are solvable in subquadratic time?"* Technically, the AYZ algorithm makes use of properties of cycles in their result and of "heavy and light" nodes (high degree and low degree, respectively). We show that degree-uniformization is a generalization of this method, and that it allows us to derive subquadratic algorithms for a larger family of joins. We devise a procedure to upper bound the processing time of a join, and an algorithm to match this upper bound. Our procedure improves the runtime exponent x relative to existing work, for a large family of joins. Moreover, for a class of graphs that we call 1-series-parallel graphs,[2] we completely resolve the subquadratic question in the following sense: For each 1-series-parallel graph, we can either solve it in subquadratic time, or we show that it cannot be solved subquadratically unless the 3-SUM problem [6] can be solved in subquadratic time. Note that 1-series-parallel graphs have fhw equal to 2. Hence, they can all be solved in quadratic time using existing algorithms; making our 3-SUM based lower bound tight. There is a known 3-SUM based lower bound of $N^{\frac{4}{3}}$ on triangle join processing, which only has a matching upper bound under the assumption that the matrix multiplication exponent $\omega = 2$. In contrast, our quadratic lower bound can be matched by existing algorithms without any assumptions on ω. To our knowledge, this makes it the only known tight bound on join processing time for small output sizes.

We also recover our sequential join results within the well-known GHD framework [10]. We do this using a novel notion of width, which we call m-width, that is no larger than fhw, and sometimes smaller than submodular width [12, 15]. While we resolve the subquadratic problem on 1-series-parallel graphs, the general subquadratic problem remains open. In the full version [12], we show that known notions of widths, such as submodular width and m-width do not fully characterize subquadratically solvable joins.

Joins on MapReduce: Degree information can also be used to improve the efficiency of joins on MapReduce. Previous work by Beame et al. [8] uses knowledge of heavy hitters (values with high degree) to improve parallel join processing on skewed data. It allows a limited range of parallelism (number of processors $p \leq \sqrt{\text{IN}}$), but subject to that achieves optimal communication for 1-round MapReduce algorithms. We use degree information to allow all levels of parallelism ($p \geq 1$) while processing the join. We also obtain an improved degree-based upper bound on output size that can be significantly better than the

[2] A 1-series-parallel graph consists of a source vertex s, a target vertex t, and a set of paths of any length from s to t, which do not share any nodes other than s and t.

AGM bound even on simple queries. Our improved parallel algorithm takes three rounds of MapReduce, matches our improved bound, and out-performs the optimal 1-round algorithm in several cases. As an example, our improved bound lets us correctly upper bound the output of a sparse triangle join (where each value has degree $O(1)$) by IN instead of $\text{IN}^{\frac{3}{2}}$ as suggested by the AGM bound. Moreover, we can process the join at maximum levels of parallelism (with each processor handling only $O(1)$ tuples) at a total communication cost of $O(\text{IN})$; in contrast to previous work which requires $\theta(\text{IN}^{\frac{3}{2}})$ communication. Furthermore, previous work [8] uses edge packings to bound the communication cost of processing a join. Edge packings have the paradoxical property that adding information on the size of subrelations by adding the subrelations into the join can make the communication cost larger. As an example suppose a join has a relation R, with an attribute A in its schema. Adding $\pi_A(R)$ to the set of relations to be joined does not change the join output. However, adding a weight term for subrelation $\pi_A(R)$ in the edge packing linear program increases its communication cost bound. In contrast, if we add $\pi_A(R)$ into the join, our degree based bound does not increase, and will in fact decrease if $|\pi_A(R)|$ is small enough.

Computing Degree Information: In some cases, degree information is not available beforehand or is out of date. In such a case, we show a simple way to compute the degrees of all values in time linear in the input size. Moreover, the degree computation procedure can be fully parallelized in MapReduce. Even after including the complexity of computing degrees, our algorithms outperform state of the art join algorithms.

Our paper is structured as follows:

- In Section 2, we describe related work.
- In Section 3, we describe a process called *degree-uniformization*, which mitigates skew. We show the MO bound on join output size that strengthens the exponent in the AGM bound, and describe a method to compute the degrees of all attributes in all relations.
- In Section 4, we present DARTS, our sequential algorithm that achieves tighter runtime exponents than state-of-the-art. We use DARTs to process several joins in subquadratic time. Then we establish a quadratic runtime lower bound for a certain class of queries modulo the 3-SUM problem. Finally we recover the results of DARTS within the familiar GHD framework, using a novel notion of width (m-width) that is tighter than fhw.
- In Section 5, we present another bound with a tighter exponent than AGM (the DBP bound), and a tunable parallel algorithm whose communication cost at maximum parallelism equals the input size plus the DBP bound. The algorithm's guarantees work on all inputs independent of skew.

2 Related Work

We divide related work into four broad categories.

New join algorithms and implementation: The AGM bound [5] is tight on the output size of a multiway join in terms of the query structure and sizes of relations in the query. Several existing join algorithms, such as NPRR [16], LFTJ [18], and Generic Join [17], have worst case runtime equal to this bound. However, there exist instances of relations where the output size is significantly smaller than the worst-case output size (given by the AGM bound), and the above algorithms can have a higher cost than the output size. We demonstrate a bound on output size that has a tighter exponent than the AGM bound by taking into account information on degrees of values, and match it with a parallelizable algorithm.

On α-acyclic queries, Yannakakis' algorithm [19] is instance optimal up to a constant multiplicative factor. That is, its cost is $O(\text{IN} + \text{OUT})$ where IN is the input size. For cyclic queries, we can combine Yannakakis' algorithm with the worst-case optimal algorithms like NPRR to get a better performance than that of NPRR alone. This is done using Generalized Hypertree decompositions (GHDS) [9, 10] of the query to answer the query in time $O(\text{IN}^{\text{fhw}} + \text{OUT})$ where fhw is a measure of cyclicity of the query. A query is α-acyclic if and only if its fhw is one. Our work allows us to obtain a tighter runtime exponent than fhw by dealing with values of different degrees separately.

Parallel join algorithms: The Shares [2] algorithm is the optimal one round algorithm for skew free databases, matching the lower bound of Beame et al. [7]. But its communication cost can be much worse than optimal when skew is present. Beame's work [8] deals with skew and is optimal among 1-round algorithms when skew is present. The GYM [1] algorithm shows that allowing $\log(n)$ rounds of MapReduce instead of just one round can significantly reduce cost. Allowing n rounds can reduce it even further. Our work shows that merely going from one to three rounds can by itself significantly improve on existing 1-round algorithms. Our parallel algorithm can be incorporated into Step 1 of GYM as well, thereby reducing its communication cost.

Using Database Statistics: The cycle detection algorithm by Alon, Yuster and Zwick [4] can improve on the fhw bound by using degree information in a sequential setting. Specifically, the fhw of a cycle is two but the AYZ algorithm [4] can process a cycle join in time $O(\text{IN}^{2-\epsilon} + \text{OUT})$ where $\epsilon > 0$ is a function of the cycle length. We generalize this, obtaining subquadratic runtime for a larger family of graphs, and develop a general procedure for upper bounding the cost of a join by dealing with different degree values separately.

Beame et al.'s work [8] also uses degree information for parallel join processing. Specifically, it assumes that all heavy hitters (values with high degree) and their degrees are known beforehand, and processes them separately to get optimal 1-round results. Their work uses edge packings to bound the cost of their algorithm. Edge packings have the counterintuitive property that adding more constraints, or more information on subrelation sizes, can worsen the edge packing cost. This suggests that edge packings alone do not provide the right framework for taking degree information into account. Our work remedies this, and the performance of our algorithm improves when more constraints are added. In addition, Beame et al. [8] assume that $M > p^2$ where M is relation size and p is the number of processors. Thus, their algorithm cannot be maximally parallelized. In contrast, our algorithm can work at all levels of parallelism, ranging from one in which each processor gets only $O(1)$ tuples to one in which a single processor does all the processing.

Degree Uniformization: The partitioning technique of Alon et al. [3] is similar to our *degree-uniformization* technique, but has stronger guarantees at a higher cost. It splits a relation into 'parts' where the maximum degree of any attribute set A in each part P is within a constant factor of the average degree of A in P. In contrast, degree-uniformization lets us upper bound the maximum degree of A in P in absolute terms, but not relative to the average degree of A in P.

Marx's work [15] uses a stronger partitioning technique to fully characterize the fixed-parameter tractability of joins in terms of the *submodular width* of their hypergraphs. Marx achieves degree-uniformity within all small projections of the output, while we only achieve uniform degrees within relations. Marx's preprocessing is expensive; the technique as written

in Section 4 of his paper [15] takes time $\Omega(\text{IN}^{2c})$ where c is the submodular width of the join hypergraph. This preprocessing is potentially more expensive than the join processing itself. Our algorithms run in time $O(\text{IN}^{\mathsf{MW}})$ with $\mathsf{MW} < c$ for several joins. Marx did not attempt to minimize this exponent, as his application was concerned with fixed parameter tractability. We were unable to find an easy way to achieve $O(\text{IN}^c)$ runtime for Marx's technique.

3 Degree Uniformization

We describe our algorithms for degree-uniformization and counting, as well as our improved output size bound. Section 3.1 introduces our notation. Section 3.2 gives a high-level overview of our join algorithms. Then, we describe the degree-uniformization which is a key step in our algorithms. In Section 3.3, we describe the MO bound, an upper bound on join output size that has a tighter exponent than the AGM bound. We provide realistic examples in which the MO bound is much tighter than the AGM bound. Finally, in Section 3.4 we describe a linear time algorithm for computing degrees.

3.1 Preliminaries and Notation

Throughout the paper we consider a multiway join. Let $\mathcal{R}$ be the set of relations in the join and $\mathcal{A}$ be the set of all attributes in those relations' schemas. For any relation R, we let $\mathsf{attr}(R)$ denote the set of attributes in the schema of R. We wish to process the join $\bowtie_{R\in\mathcal{R}} R$, defined as the set of tuples t such that $\forall R \in \mathcal{R} : \pi_{\mathsf{attr}(R)}(t) \in R$. $|R|$ denotes the number of tuples in relation R. For any set of attributes $A \subseteq \mathcal{A}$, a *value* in attribute set A is defined as a tuple from $\bigcup_{R\in\mathcal{R}:A\subseteq\mathsf{attr}(R)} \pi_A(R)$. For any $A \subseteq \mathsf{attr}(R)$, the *degree* of a value v in A in relation R is given by the number of times v occurs in R i.e. $\mathsf{deg}(v, R, A) = |\{t \in R \mid \pi_A(t) = v\}|$. For all values v of A in R, we must have $\mathsf{deg}(v, R, A) \geq 1$.

In Section 4, we denote a join query with a hypergraph G; the vertices in the graph correspond to attributes and the hyperedges to relations. We use $R(X_1, X_2, \ldots, X_k)$ to denote a relation R having schema $(X_1, X_2, \ldots, X_k)$. IN denotes the input size i.e. sum of sizes of input relations, while OUT denotes the output size. Our output size bounds, computation costs, and communication costs will be expressed using O notation which hides polylogarithmic factors i.e. $\log^c(\text{IN})$, for some c not dependent on number of tuples IN (but possibly dependent on the number of relations/attributes). All ensuing logarithms in the paper, unless otherwise specified, will be to the base IN.

AGM Bound: Consider the following linear program:

▶ **Linear Program 1.**

$$\text{Minimize} \sum_{R\in\mathcal{R}} w_R \log(|R|) \text{ such that } \forall a \in \mathcal{A} : \sum_{R\in\mathcal{R}:a\in\mathsf{attr}(R)} w_R \geq 1$$

A valid assignment of weights w_R to relation R in the linear program is called a *fractional cover*. If $\rho*$ is the minimum value of the objective function, then the AGM bound on the join output size is given by $\text{IN}^{\rho*}$. In general, for any set of relations $\mathcal{R}$, we use $\mathsf{AGM}(\mathcal{R})$ to denote the AGM bound on $\bowtie_{R\in\mathcal{R}} R$.

3.2 Degree Uniformization

We describe our high level join procedure in Algorithm 1. In Step 1, we compute the degree of each value in each attribute set A, in each relation R. If the degrees are available beforehand,

Algorithm 1: High level join algorithm

Input: Set of relations $\mathcal{R}$, Bucket range parameter L
Output: $\bowtie_{R\in\mathcal{R}} R$
1. Compute $\mathsf{deg}(v, R, A)$ for each $R \in \mathcal{R}, A \subseteq \mathsf{attr}(R), v \in \pi_A(R)$
2. Compute the set of all L-degree configurations $\mathcal{C}_L$
foreach $c \in \mathcal{C}_L$ **do**
  3.1. Compute partition $R(c)$ of each relation R
  3.2. Compute $\mathcal{R}(c) = \{R(c) \mid R \in \mathcal{R}\}$
  4. Compute join $J_c = \bowtie_{R\in\mathcal{R}(c)} R$
5. **return** $\bigcup_{c\in\mathcal{C}_L} J_c$

due to being maintained by the database, then we can skip this step. We further describe this step in Section 3.4.

Steps 2, 3 together constitute *degree-uniformization*. In these steps, we partition each relation R by degree. In particular, we assign each value in a relation to a bucket based on its degree: with one bucket for degrees in $[1, L)$, one for degrees in $[L, L^2)$, and so on. Then we process the join using one partition from each relation, for all possible combinations of partitions. Each such combination is referred to as a *degree configuration*. We use c to denote any individual degree configuration, $\mathcal{C}_L$ to denote the set of all degree configurations, $R(c)$ to denote the part of relation R being joined in configuration c, and $\mathcal{R}(c)$ to denote $\{R(c) \mid R \in \mathcal{R}\}$. Step 2 consists of enumerating all degree configurations, and Step 3 consists of finding the partition of each relation corresponding to each degree configuration.

In Step 4, we compute $J_c = \bowtie_{R\in\mathcal{R}(c)} R$ for each degree configuration c. Section 4 describes how to perform Step 4 in a sequential setting, while Section 5 describes it for a MapReduce setting. Step 5 combines the join outputs for each c to get the final output.

Steps 1, 2, 3 and 5 can be performed efficiently in MapReduce as well as sequential settings; thus the cost of Algorithm 1 is determined by Step 4. Step 4 is carried out differently in sequential and MapReduce settings. Its cost in the sequential setting is lower than the cost in a MapReduce setting. Steps 1, 2, and 3 have a cost of $O(\text{IN})$, while Step 5 has cost $O(\text{OUT})$. Since reading the input and output always has a cost of $O(\text{IN} + \text{OUT})$, the only extra costs we incur are in Step 4 when we actually process the join. Costs for Step 4 will be described in Sections 4 and 5.

Degree-uniformization: Now we describe degree-uniformization in detail. We pick a value for a parameter L which we call 'bucket range', and define buckets $B_l = [L^l, L^{l+1})$ for all $l \in \mathbb{N}$. Let $\mathcal{B} = \{B_0, B_1, \ldots, \}$. For any two buckets $B_i, B_j \in \mathcal{B}$, we say $B_i \leq B_j$ iff $i \leq j$. A degree configuration specifies a unique bucket for each relation and set of attributes in that relation. Formally:

▶ **Definition 1.** Given a parameter L, we define a degree configuration c to be a function that maps each pair (R, A) with $R \in \mathcal{R}, A \subseteq \mathsf{attr}(R)$ to a unique bucket in $\mathcal{B}$ denoted $c(R, A)$, such that

$$\forall R, A, A' : A' \subseteq A \subseteq \mathsf{attr}(R) \Rightarrow c(R, A) \leq c(R, A')$$

$$\forall R : c(R, \mathsf{attr}(R)) = B_0 \text{ and } c(R, \emptyset) = B_{\lfloor \log_L(|R|)\rfloor}$$

▶ **Example 2.** If a join has relations $R_1(X, Y), R_2(Y)$, then a possible configuration is $(R_1, \emptyset) \mapsto B_3$, $(R_1, \{X\}) \mapsto B_1$, $(R_1, \{Y\}) \mapsto B_2$, $(R_1, \{X, Y\}) \mapsto B_0$, $(R_2, \emptyset) \mapsto B_1$, $(R_2, \{Y\}) \mapsto B_0$.

▶ **Definition 3.** Given a degree configuration c for a given L, and a relation $R \in \mathcal{R}$, we define $R(c)$ to be the set of tuples in R that have degrees consistent with c. Specifically:

$$R(c) = \{t \in R \mid \forall A \subseteq \mathsf{attr}(R) : \mathsf{deg}(\pi_A(t), R, A) \in c(R, A)\} .$$

We define $\mathcal{C}_L$ to be the set of all degree configurations with parameter L.

▶ **Example 4.** For a tuple $(a, b) \in R$, where $L^2 \leq |R| < L^3$, with the degree of a in B_1, and that of b in B_2, the tuple would be in $R(c)$ if $c(R, \emptyset) = B_2, c(R, \{A\}) = B_1, c(R, \{B\}) = B_2, c(R, \{A, B\}) = B_0$. On the other hand, it would not be in $R(c)$ if $c(R, \{A\}) = B_0$, even if we had $c(R, \{A, B\}) = B_0$, $c(R, \{B\}) = B_2$.

A degree configuration also bounds degrees of values in sub-relations, as stated below:

▶ **Lemma 5.** *For all* $R \in \mathcal{R}, A' \subseteq A \subset \mathsf{attr}(R), L > 1, c \in \mathcal{C}_L, v \in \pi_{A'}(R), j \geq i \geq 0$*:*

$$c(R, A) = B_i \wedge c(R, A') = B_j \Rightarrow \mathsf{deg}(v, \pi_A(R(c)), A') \leq L^{j+1-i} .$$

Choosing L: The optimal value of parameter L depends on our application. L has three effects : (i) For the DBP/MO bounds (Sections 3.3, 5) and sequential algorithm (Section 4), the error in output size estimates is exponential in L (with the exponent depending only on the number of attributes) (ii) The load per processor for the parallel algorithm (Section 5) is $O(L)$ (iii) the number of rounds for the parallel algorithm is $\log_L(\text{IN})$. As a result, we choose a small $L(= 2)$ for the sequential algorithm and DBP/MO bounds, and a larger L $(=$ load capacity $= \text{IN}^\gamma$ for some $\gamma < 1)$ for the parallel algorithm.

3.3 Beyond AGM: The MO Bound

We now use degree-uniformization to tighten our upper bound on join output size.

▶ **Definition 6.** Let $\mathcal{R}$ be a set of relations, with attributes in $\mathcal{A}$. For each $R \in \mathcal{R}, A \subseteq \mathsf{attr}(R)$, let $d_{R,A} = \max_{v \in \pi_A(R)} \mathsf{deg}(v, R, A)$. If $A = \emptyset$ then $d_{R,\emptyset} = |R|$. And for any $A \subseteq B \subseteq \mathsf{attr}(R)$, let $d(A, B, R)$ denote $\log(d_{\pi_B(R),A})$. Then consider the following linear program for L.

▶ **Linear Program 2.**

$$\text{Maximize } s_{\mathcal{A}} \text{ s. t. (i) } s_\emptyset = 0 \text{ (ii) } \forall A, B \text{ s.t. } A \subseteq B : s_A \leq s_B$$
$$\text{(iii) } \forall A, B, E, R \text{ s.t. } R \in \mathcal{R}, E \subseteq \mathcal{A}, A \subseteq B \subseteq \mathsf{attr}(R) : s_{B \cup E} \leq s_{A \cup E} + d(A, B, R)$$

We define $m_{\mathcal{A}}$ to be the maximum objective value of the above program.

▶ **Proposition 7.** The output size $\bowtie_{R \in \mathcal{R}} R$ is in $O(\text{IN}^{m_{\mathcal{A}}})$.

Intuitively, for any $A \subseteq \mathcal{A}$, s_A stands for possible values of $\log(|\pi_A(\bowtie_{R \in \mathcal{R}} R)|)$. This explains the first two constraints (projecting onto the empty set gives size 1, and the projection size over A is monotone in A). For the third constraint, we use the fact that each value in A has at most $\text{IN}^{d(A,B,R)}$ values in B, thus each tuple in $\pi_{A \cup E}(\bowtie_{R \in \mathcal{R}} R)$ can give us at most $\text{IN}^{d(A,B,R)}$ tuples in $\pi_{B \cup E}(\bowtie_{R \in \mathcal{R}} R)$. The linear program attempts to maximize the total output size $(\text{IN}^{s_{\mathcal{A}}})$ while still satisfying the constraints.

We now define the MO bound.

▶ **Definition 8.** Let $\mathsf{MO}(\mathcal{R})$ denote the value $m_{\mathcal{A}}$ for any join query consisting of relations $\mathcal{R}$. Then the MO bound is given by $\sum_{c \in \mathcal{C}_2} \text{IN}^{\mathsf{MO}(\mathcal{R}(c))}$.

▶ **Theorem 9.** *The* MO *bound is in* $O(AGM(\mathcal{R}))$.

The constant in the $O()$ notation depends on the number of attributes in the query, but not on the number of tuples. This result is proved in two steps. Theorem 26 states that the DBP bound (introduced in Section 5) is smaller than the AGM bound, while Theorem 23 implies that the MO bound is smaller than the DBP bound times a constant.

▶ **Example 10.** Let $L = 2$ for this example. Consider a triangle join $R(X,Y) \bowtie S(Y,Z) \bowtie T(Z,X)$. Let $|R| = |S| = |T| = N$. The AGM bound on this is $N^{3/2}$. Let the degree of each value x in X in both R and T be h. For different values of h we will find an upper bound on $m_{\{X,Y,Z\}}$ and hence on the output size.

Case 1. $h < \sqrt{N}$**:** Then $s_{\{X\}} \leq s_{\emptyset} + d(\emptyset, \{X\}, R) = \log(N/h)$. Thus, $s_{\{X,Y\}} \leq s_{\{X\}} + d(\{X\}, \{X,Y\}, R) \leq \log(N/h) + \log(h) = \log(N)$. Finally, $s_{\{X,Y,Z\}} \leq s_{\{X,Y\}} + d(\{X\}, \{X,Z\}, T) \leq \log(N) + \log(h)$. Thus the MO bound is $\leq Nh < N^{3/2}$.

Case 2. $h > \sqrt{N}$**:** Since there can be at most N/h distinct X values, we have $d(\{Y\}, \{X,Y\}, R) \leq \log(N/h))$. More if the degree of Y in S in a degree configuration is g, then $s_{\{Y,Z\}} \leq s_{\{Y\}} + d(\{Y\}, \{Y,Z\}, S) \leq \log(N/g) + \log(g) = \log(N)$. Finally, $s_{\{X,Y,Z\}} \leq s_{\{Y,Z\}} + d(\{Y\}, \{X,Y\}, R) \leq \log(N) + \log(N/h) = \log(N^2/h) < N^{3/2}$.

The MO bound has a strictly smaller exponent than AGM unless $h \approx \sqrt{N}$. Computing the AGM bound individually over each degree configuration does not help us do better, as the above example can have all tuples in a single degree configuration.

▶ **Example 11.** Consider a matching database [7], where each attribute has the same domain of size N, and each relation is a matching. Thus each value has degree 1, and $d(A, B, R)$ equals 0 when $A \neq \emptyset$ and 1 if $A = \emptyset$. The MO bound on such a database trivially equals N, which can have an unboundedly smaller exponent than the AGM bound.

The full version similarly compares the DBP and AGM bounds, showing that DBP (and hence MO) has a strictly smaller exponent than AGM for 'almost all' degrees.

3.4 Degree Computation

If we do not know degrees in advance we can compute them on the fly, as stated below:

▶ **Lemma 12.** *Given a relation* R, $A \subseteq \mathsf{attr}(R)$, *and* $L > 1$, *we can find* $\mathsf{deg}(v, R, A)$ *for each* $v \in \pi_A(R)$ *in a MapReduce setting, with* $O(|R|)$ *total communication, in* $O(\log_L(|R|))$ *MapReduce rounds, and at* $O(L)$ *load per processor. In a sequential setting, we can compute degrees in time* $O(|R|)$.

To perform degree-uniformization, we compute degrees for all relations R, and all $A \subseteq \mathsf{attr}(R)$. The number of such (R, A) pairs is exponential in the number and size of relations, but is still constant with respect to the input size IN.

4 Sequential Join Processing

We present our results on sequential join processing. Section 4.1 describes our problem setting. In Section 4.2 we present our sequential join algorithm, *DARTS* (for **D**egree-based **A**ttribute-**R**elation **T**ransform**s**). DARTS handles queries consisting of a join followed by a projection. A join alone is simply a join followed by projection onto all attributes. We

pre-process the input by performing degree-uniformization, and then run DARTS on each degree configuration. DARTS works by performing a sequence of *transforms* on the join problem; each transform reduces the problem to smaller problems with fewer attributes or relations. We describe each of the transforms in turn. We then show that DARTS can be used to recover (while potentially improving on) known join results such as those of the NPRR algorithm, Yannakakis' algorithm, the fhw algorithm, and the AYZ algorithm.

In Section 4.3, we apply DARTS to the subquadratic joins problem; presenting cases in which we can go beyond existing results in terms of the runtime exponent. For a family of joins called 1-series-parallel graphs, we obtain a full dichotomy for the subquadratic joins problem. That is, for each 1-series-parallel graph, we can either show that DARTS processes its join in subquadratic time, or that no algorithm can process it in subquadratic time modulo the 3-SUM problem. Note that 1-series-parallel graphs have treewidth 2, making them easily solvable in quadratic time. Thus, our 3-SUM based quadratic lower bound on some of the graphs is tight making it, to our knowledge, the only tight bound for join processing time with small output sizes. In contrast, there is a $N^{\frac{4}{3}}$ lower bound (using 3-SUM) for triangle joins, but its matching upper bound depends on the additional assumption that the matrix multiplication exponent equals two.

In Section 4.4, we show that most results of the DARTS algorithms can be recovered using the well known framework of Generalized Hypertree Decompositions (GHDs), along with a novel notion of width we call m-width. m-width is no larger than fhw, and sometimes smaller than submodular width [12].

4.1 Setting

In this section, we focus on a sequential join processing setting. We are especially interested in the subquadratic joins problem stated below:

▶ **Problem 1.** For any graph G, we let each node in the graph represent an attribute and each edge represent a relation of size N. Then we want to know, for what graphs G can we process a join over the relations in *subquadratic* time, i.e. $O(N^{2-\epsilon} + \text{OUT})$ for some $\epsilon > 0$?

Performing a join in subquadratic time is especially important when we have large datasets being joined, and the output size is significantly smaller than the worst case output size. Note that we define subquadratic to be a poly(N) factor smaller than N^2, so for instance a $\frac{N^2}{\log N}$ algorithm is not subquadratic by our definition.

As an example, if a join query is α-acyclic, then Yannakakis' algorithm can answer it in time $O(N + \text{OUT})$, which is subquadratic. More generally, if the fractional hypertree width (fhw) of a query is $\rho*$, the join can be processed in time $O(N^{\rho*} + \text{OUT})$ using a combination of the NPRR and Yannakakis' algorithms. The fhw of an α-acyclic query is one. For any graph with $\mathsf{fhw} < 2$, we can process its join in subquadratic time. The AYZ algorithm allows us to process joins over length n cycles in time $O(N^{2-\frac{1}{1+\lceil \frac{n}{2} \rceil}} + \text{OUT})$, even though cycles of length ≥ 4 have $\mathsf{fhw} = 2$. To the best of our knowledge, this is the only previous result that can process a join with fhw ≥ 2 in subquadratic time.

The DARTS algorithm is applicable to any join-project problem and not just those with equal relation sizes like in Problem 1. Applying DARTS to Problem 1 lets us process several joins in subquadratic time despite having fhw ≥ 2. Section 4.4 recovers the subquadratic runtimes of DARTS using GHDs that have m-width < 2.

4.2 The DARTS algorithm

We now describe the DARTS algorithm. The problem that DARTS solves is more general than a join. It takes as input a set of relations $\mathcal{R}$, and a set of attributes $\mathcal{O}$ (which stands for **O**utput), and computes $\pi_{\mathcal{O}} \bowtie_{R\in\mathcal{R}} R$. When $\mathcal{O} = \mathcal{A}$, the problem reduces to just a join. We first pre-process the inputs by performing degree-uniformization. Then each degree configuration is processed separately by DARTS. The L parameter for degree-uniformization is set to be very small ($O(1)$). The total computation time is the sum of the computation times over all degree configurations. Let $G = (c, \mathcal{R}(c), \mathcal{O})$. That is, G specifies the query relations, output attributes, and degrees for each attribute set in each relation according to the degree configuration. We let $c_G, \mathcal{R}_G, \mathcal{O}_G$ denote to degree configuration of G, the relations in G, and the output attributes of G. We define two notions of runtime complexity for the join-project problem on G:

▶ **Definition 13.** $Q(G)$ is the smallest value such that a join-projection with query structure, degrees, and output attributes given by those in G can be processed in time $O(Q(G) + \text{OUT})$. $P(G)$ is the smallest value such that a join-projection with query structure, degrees, and output attributes given by those in G can be processed in time $O(P(G))$.

▶ **Example 14.** As an example of the difference between P and Q, consider a chain join G with relations $R_1(X_1, X_2)$, $R_2(X_2, X_3)$, $R_3(X_3, X_4)$, and $\mathcal{O} = \{X_1, X_2, X_3, X_4\}$. All relations have size N, and the degree of each attribute in each relation is $\sqrt{N}$. Then $P(G)$ would be N^2, the worst case size of the output (where all attributes have $\sqrt{N}$ values and each relation is a full cartesian product). $Q(G)$ on the other hand would be N because the join is α-acyclic, and Yannakakis' algorithm lets us process the join in time $O(N + \text{OUT})$.

4.2.1 Heavy, Light and Split

The DARTS algorithm performs a series of *transforms* on G, each of which reduces it to a smaller problem. In each step, it chooses one of three types of transforms, which we call *Heavy*, *Light* and *Split*. Each transform takes as input G itself and either an attribute or a set of attributes in the relations of G. Then it reduces the join-project problem on G to a simpler problem via a *procedure*. This reduction gives us a *bound* on $P(G)$ and/or $Q(G)$ in terms of the P and Q values of simpler problems. We describe each of these transforms in turn, along with their input, procedure, and bound.

Heavy

Input: G, An attribute X

Procedure: Let $\mathcal{R}_X = \{R \in \mathcal{R}(c) \mid X \in \mathsf{attr}(R)\}$. Then we compute the values of $x \in X$ that lie in all relations in $\mathcal{R}_X$ i.e. $\mathsf{vals}(X) = \bigcap_{R\in\mathcal{R}_X} \pi_X R$. Then for each $x \in \mathsf{vals}(X)$, we marginalize on x. That is, we solve the *reduced problem*:

$$J_x = \pi_{\mathcal{O}\setminus\{X\}} \left(\bowtie_{R\in(\mathcal{R}(c)\setminus\mathcal{R}_X)} R \bowtie_{R\in\mathcal{R}_X} (\pi_{\mathcal{A}\setminus\{X\}}\sigma_{X=x}R)\right) .$$

Our final output is $\bigcup_{x\in\mathsf{vals}(X)}(\pi_{\mathcal{O}}x) \times J_x$. For each relation $R \in \mathcal{R}_X$, let d_R be the maximum value in bucket $c(R, \{X\})$. So $|\mathsf{vals}(X)| \leq \min_{R\in\mathcal{R}_X} \frac{|R|}{d_R}$. Secondly, in each reduced problem J_x, the size of each reduced relation $\pi_{\mathcal{A}\setminus\{X\}}\sigma_{X=x}R$ for $R \in \mathcal{R}_X$ reduces to at most d_R. Let G' denote the reduced relations, degrees, and output attributes for J_x. This gives us:

Bound: $Q(G) \leq \left(\min_{R\in\mathcal{R}_X} \frac{|R|}{d_R}\right) Q(G')$, $P(G) \leq \left(\min_{R\in\mathcal{R}_X} \frac{|R|}{d_R}\right) P(G')$

Light

Input: G, An attribute set X

Procedure: The light transform reduces the number of relations in G. Define $\mathcal{R}_X = \{R \in \mathcal{R}(c) \mid \mathsf{attr}(R) \subseteq X\}$. We compute $R_X = \bowtie_{R \in \mathcal{R}(c)} \pi_X R$. This subjoin is computed using a sequential version of the parallel technique in Section 5. Hence it takes time equal to the DBP bound on that join. Then we delete relations in $\mathcal{R}_X$ from G, and add R_X into $\mathcal{R}_G$. The degrees for attributes in R_X can be computed in terms of degrees in the relations from $\mathcal{R}_X$. As long as $|\mathcal{R}_X| > 1$, this gives us a reduced problem G'. $\mathcal{O}$ stays unchanged for the reduced problem. The size of relation R_X can be upper bounded using the DBP bound as well. Let $\mathsf{DBP}(G, X)$ denote this bound.

Bound: $Q(G) \leq \mathsf{DBP}(G, X) + Q(G')$, $P(G) \leq \mathsf{DBP}(G, X) + P(G')$

Split

Input: G, An *articulation set* S of attributes [11] such that there are joins G_1, G_2 whose attribute sets have no attribute outside S in common, and $\mathcal{R}_G \subseteq \mathcal{R}_{G_1} \cup \mathcal{R}_{G_2}$. Also, S satisfies either (i) $S \subseteq \mathcal{O}$, or (ii) $\mathcal{O} \subseteq \bigcup_{R \in \mathcal{R}_{G_2}} \mathsf{attr}(R)$.

Procedure: We compute $R_S = \pi_S \left(\bowtie_{R \in \mathcal{R}_{G_1}} R\right)$. This takes time $P(G_1')$, where G_1' is like G_1 but with $\mathcal{O}_{G_1'} = S$. Let $J_2 = \left(\bowtie_{R \in \mathcal{R}_{G_2}} R\right) \bowtie R_S$. If $\mathcal{O} \subseteq \bigcup_{R \in \mathcal{R}_{G_2}} \mathsf{attr}(R)$, then we compute and output $\pi_{\mathcal{O}} J_2$, and we are done. This step costs $P(G_2)$. Otherwise, $S \subseteq \mathcal{O}$. We compute $O_2 = \pi_{\mathcal{O}} J_2$. Each tuple in O_2 has a matching output tuple for G. Then we set $R_S = R_S \cap \pi_S O_2$ and compute $O_1 = \pi_{\mathcal{O}}(\bowtie_{R \in \mathcal{R}_{G_1}} R \bowtie R_S)$. Then for each tuple $t \in R_S$, we take each pair of matching tuples $t_1 \in O_1$, $t_2 \in O_2$ and output $t_1 \bowtie t_2$. Let G_1'' be like G_1, but with $\mathcal{O}_{G_1''} = \mathcal{O} \cap \left(\bigcup_{R \in \mathcal{R}_{G_1}} \mathsf{attr}(R)\right)$, and G_2'' be defined similarly. This gives us:

Bound: If $S \subseteq \mathcal{O}$, then $Q(G) \leq P(G_1') + Q(G_1'') + Q(G_2'')$. If $\mathcal{O} \subseteq \bigcup_{R \in \mathcal{R}_{G_2}} \mathsf{attr}(R)$, then $P(G) \leq P(G_1') + P(G_2)$.

4.2.2 Combining the Transforms

Once we know the transforms, the DARTS algorithm is quite straightforward. It considers all possible sequences of transforms that can be used to solve the problem, and picks the one that gives the smallest upper bound on $Q(G)$. The number of such transform sequences is exponential in the number of attributes and relations, but constant with respect to data size. The P and Q values of various Gs can be computed recursively given a degree configuration. The G' obtained in each recursive step itself specifies a degree configuration, over a smaller problem. The degrees in G' can be computed in terms of degrees in G. Note that in some cases, we do not have cost bounds available e.g. we do not have a P bound for the Split transform when $S \subseteq \mathcal{O}$. This is a part of the DARTS algorithm. DARTS only considers performing a transform when it can upper bound the resulting cost.

We show that DARTS can be used to recover existing results on sequential joins.

▶ **Proposition 15.** If we compute the join using a single Light transform, our total cost is $\leq$ the AGM bound, thus recovering the result of the NPRR algorithm [16].

▶ **Proposition 16.** If we successively apply the Split transform on an α-acyclic join, with G_1 being an ear of the join in each step, then the total cost of our algorithm becomes $O(\text{IN} + \text{OUT})$, recovering the result of Yannakakis' algorithm [19].

▶ **Proposition 17.** If a query has fractional hypertree width equal to fhw, then using a combination of Split and Light transforms, we can bound the cost of running DARTS by $O(\text{IN}^{fhw} + \text{OUT})$, recovering the fractional hypertree width result.

▶ **Proposition 18.** A cycle join of length n with all relations having size N, can be processed by DARTS in time $O(N^{2-\frac{1}{1+\lceil \frac{n}{2} \rceil}} + \text{OUT})$, recovering the result of the AYZ algorithm [4].

In the next subsection, we present a few of the cases in which we can go *beyond* existing results. Since we are primarily interested in joins, the output attribute set $\mathcal{O}$ below is always assumed to be $\mathcal{A}$.

4.3 Subquadratic Joins

Now we consider applications of DARTS to the subquadratic joins problem. Analyzing a run of DARTS on a join graph allows us to obtain a subquadratic runtime upper bound in several cases. We now define a set of graphs for which we have a complete decision procedure to determine if they can be solved in subquadratic time modulo the 3-SUM problem.

1-series-parallel graphs

▶ **Definition 19.** A 1-series-parallel graph is one that consists of :

- A source node X_S
- A sink node X_T
- Any number of paths, of arbitrary length, from X_S to X_T, having no other nodes in common with each other

Equivalently, a 1-series-parallel graph is a series parallel graph that can be obtained using any number of series transforms (which creates paths) followed by exactly one parallel transform, which joins the paths at the endpoints. A cycle is a special case of a 1-series-parallel graph.

▶ **Theorem 20.** *For 1-series-parallel graphs, the following decision procedure determines whether or not the join over that graph can be processed in sub-quadratic time:*

1. *If there is a direct edge (path of length one) between X_S and X_T, then the join can be processed in sub-quadratic time. Else:*
2. *Remove all paths of length two between X_S and X_T, as they do not affect the sub-quadratic solvability of the join problem. Then*
3. *If the remaining number of paths (obviously all having length ≥ 3) is ≥ 3, then the join cannot be processed in subquadratic time (modulo 3-SUM). If the number of remaining paths is < 3, then the graph can be solved in sub-quadratic time.*

Theorem 20 establishes the decision procedure for subquadratic solvability of 1-series-parallel graphs. The full version gives an example of a subquadratic solution for a specific 1-series-parallel graph, namely $K_{2,n}$, followed by an example on the general bipartite graph $K_{m,n}$. In both these examples, DARTS achieves a better runtime exponent than previously known algorithms. We now make three statements that together imply Theorem 20 (formally stated and proved in the full version).

- If we have a 1-series-parallel graph, which has a direct edge from X_S to X_T (i.e. a path of length 1), then a join on that graph can be processed in subquadratic time.
- Suppose we have a 1-series-parallel graph G, which does not have a direct edge from X_S to X_T, but has a vertex X_U such that there is an edge from X_S to X_U and from X_U to X_T (i.e. a path of length 2 from X_S to X_T). Let G' be the graph obtained by deleting

the vertex X_U and edges $X_S X_U$ and $X_U X_T$. Then the join on G can be processed in subquadratic time if and only if that on G' can be processed in subquadratic time.

- Let G be any 1-series-parallel graph which does not have an edge from X_S to X_T, but has ≥ 3 paths of length at ≥ 3 each, from X_S to X_T. Then a join over G can be processed in subquadratic time only if the 3-SUM problem can be solved in subquadratic time.

4.4 A new notion of width (m-width)

We demonstrate a way to formulate the DARTS algorithm for joins in terms of GHDs.

For each $A \in \mathcal{A}$, we define m_A similarly to how we defined $m_{\mathcal{A}}$ in Section 3.3. Specifically, for each A, we use the same constraints as in linear program 2, but the objective is set to Maximize s_A instead of Maximize $s_{\mathcal{A}}$. m_A is then defined as the value of this objective function. We let $\mathsf{Prog}(A)$ denote the above linear program for finding m_A. Then the size $|\pi_A(\bowtie_{R\in\mathcal{R}} R)|$ must be bounded by IN^{m_A} for all $A \subseteq \mathcal{A}$. Moreover, for any GHD $D = (\mathcal{T}, \chi)$ of query $\mathcal{R}$, we can define $\mathsf{MW}(D, \mathcal{R})$ to be $\max_{t\in\mathcal{T}}(m_{\chi(t)})$. And $\mathsf{MW}(\mathcal{R})$ is simply the minimum value of $\mathsf{MW}(D, \mathcal{R})$ over all GHDs D. Thus we have:

▶ **Definition 21.** The m-width of a join query $\bowtie_{R\in\mathcal{R}} \mathcal{R}$ (possibly with non-uniform degrees), is given by $\max_{c\in\mathcal{C}_2} \mathsf{MW}(\mathcal{R}(c))$.

▶ **Theorem 22.** *A query with m-width* MW *can be answered in time* $O(\text{IN}^{\mathsf{MW}} + \text{OUT})$.

This theorem lets us recover all our subquadratic joins results as well. That is, for the 1-series-parallel graphs that have a subquadratic join algorithm (as per Theorem 20), we can construct a GHD that has m-width less than 2.

The MO bound is tighter than the DBP bound (and consequently, the AGM bound, as stated in Theorem 9 earlier).

▶ **Theorem 23.** *For any join query $\mathcal{R}$, and any degree configuration $c \in \mathcal{C}_2$,* $\mathsf{MO}(\mathcal{R}(c)) \leq \mathsf{DBP}(\mathcal{R}(c), 2) + |C|\log(2)$*, where C is the cover used in the DBP bound.*

Note that since logarithms are to the base IN, the $|C|\log(2)$ term is negligible even though it goes in the exponent of the bound i.e. its exponent is a constant. Theorems 22 and 23 let us recover all the results of the DARTS algorithm.

The theorems also imply that our new notion of width (m-width) is tighter than fhw. The full version [12] compares m-width to submodular width (which, barring m-width, is the tightest known notion of width applicable to general joins), showing examples where m-width is tighter than submodular width. But we do not know in general if m-width is tighter than submodular width.

The full version also shows that while m-width < 2 implies subquadratic solvability, the converse is not true; we show an example join which has m-width and submodular width $= 2$ but can be solved in subquadratic time. Thus known notions of width do not fully characterize subquadratically solvable graphs.

5 Parallel Join Processing

Like in sequential settings, degree-uniformization can be applied in a MapReduce setting. We first present the DBP bound, which is a bound on output size that is tighter than AGM bound (but not tighter than MO), and characterizes the complexity of our parallel algorithm. Then we present a 3-round MapReduce algorithm whose cost equals the DBP bound at the highest level of parallelism.

The DBP Bound

We start by defining a quantity called the Degree-based packing (DBP).

▶ **Definition 24.** Let $\mathcal{R}$ be a set of relations, with attributes in $\mathcal{A}$. Let C denote a cover i.e. a set of pairs (R, A) such that $R \in \mathcal{R}$, $A \subseteq \mathsf{attr}(R)$, and $\bigcup_{(R,A)\in C} A = \mathcal{A}$. Let $L > 1$. Then, consider the following linear program for C, L.

▶ **Linear Program 3.**

$$\text{Minimize } \sum_{a\in\mathcal{A}} v_a \text{ such that } \forall (R,A) \in C, \forall A' \subseteq A : \sum_{a\in A'} v_a \geq \log\left(\frac{d_{\pi_A(R),A\setminus A'}}{L}\right)$$

If $O_{C,L}$ is the maximum objective value of the above program, then we define $\mathsf{DBP}(\mathcal{R}, L)$ to be $\min_C O_{C,L}$ where the minimum is taken over all covers C.

▶ **Proposition 25.** Let $L > 1$ be a constant. Then the output size of $\bowtie_{R\in\mathcal{R}} R$ is in $O(\text{IN}^{\mathsf{DBP}(\mathcal{R},L)})$.

We implicitly prove this result by providing a parallel algorithm whose complexity equals the output size bound at the maximum parallelism level. We can now define the DBP bound. We arbitrarily set $L = 2$ for this definition (choosing another constant value only changes the bound by a constant factor). Thus, we define the *DBP bound* to be $\sum_{c\in\mathcal{C}_2} \text{IN}^{\mathsf{DBP}(\mathcal{R}(c),2)}$. As a simple corollary, the output size of the join is $\leq$ the DBP bound.

▶ **Theorem 26.** *For each degree configuration $c \in \mathcal{C}_L$, $\text{IN}^{\mathsf{DBP}(\mathcal{R}(c),L)} \leq \mathsf{AGM}(\mathcal{R}(c))$.*

We prove this theorem using a sequence of linear program transformations, starting with the AGM bound, and ending with the DBP bound, which each transformation decreasing the objective function value. The key transform is the fifth one, where we switch from a cover-based program to a packing-based program. We show in the full version that the DBP bound has a strictly better exponent than AGM for 'almost all' degrees.

Parallel Join Algorithm

We present our parallel 3-round join algorithm. The algorithm works at all levels of parallelism specified by load level L. Its communication cost matches the DBP bound when $L = O(1)$. We formally state the result, and then provide an example of its performance.

▶ **Theorem 27.** *For any value of L, we can process a join in $O(\log_L(\text{IN}))$ rounds (three rounds if degrees are already known) with load $O(L)$ per processor and a communication cost of $O(\text{IN} + \text{OUT} + \max_{c\in\mathcal{C}_L} L \cdot \text{IN}^{\mathsf{DBP}(\mathcal{R}(c),L)})$.*

We briefly sketch the algorithm here, and provide the full proof in the full version. We start by performing degree uniformization. Now consider any configuration c. We solve Linear Program 3 over all covers. Let C be the optimal cover, and v_a the values in the optimal solution to Linear Program 3 with cover C. We join $\bowtie_{(R,A)\in C} \pi_A(R)$ using the Shares algorithm with share IN^{v_a} for attribute a. Finally, we semijoin relations not in C with the result. The following lemma gives us our required communication cost and load bounds.

▶ **Lemma 28.** *The shares algorithm, where each attribute a has share IN^{v_a}, where v_a is from the solution to Linear Program 3, has a load of $O(L)$ per processor with high probability, and a communication cost of $O(\max_{c\in\mathcal{C}_L} L \cdot \text{IN}^{\mathsf{DBP}(\mathcal{R}(c),L)})$.*

▶ **Example 29.** Consider the sparse triangle join, with $\mathcal{R} = \{R_1(X,Y), R_2(Y,Z), R_3(Z,X)\}$. Each relation has size N, and each value has degree $O(1)$. When the load level is $L < N$, the join requires $\mathsf{DBP}(\mathcal{R}, L) = \frac{N}{L}$ processors. Equivalently, when we have p processors, the load per processor is $\frac{N}{p}$, which means it decreases as fast as possible as a function of p.

In contrast the vanilla shares algorithm allocates a share of $p^{\frac{1}{3}}$ to each attribute, and the load per processor is $Np^{-\frac{2}{3}}$. Current state of the art work [8] has a load of $Np^{-\frac{2}{3}}$ as well.

We further explore and generalize this example in the full version, where we also show an example where our parallel algorithm operating at maximum parallelism still has lower total cost than existing state-of-the-art sequential algorithms.

6 Conclusion and Future Work

We demonstrated that using degree information for a join can let us tighten the exponent of our output size bound. We presented a parallel algorithm that works at all levels of parallelism, and whose communication cost matches a tightened bound at the maximum parallelism level. We proposed the question of deciding which joins can be processed in subquadratic time, and made some progress towards answering it. We showed a tight quadratic lower bound for a family of joins, making it the only known tight bound that makes no assumptions about the matrix multiplication exponent. We presented an improved sequential algorithm, namely DARTS, that generalizes several known join algorithms, while outperforming them in several cases. We recovered the results of DARTS in the GHD framework, using a novel notion of width that is tighter than fhw and sometimes tighter than submodular width as well.

We presented several cases in which DARTS outperforms existing algorithms, in the context of subquadratic joins. However, it is likely that DARTS outperforms existing algorithms on joins having higher treewidths as well. A fuller exploration of the improved upper bounds achieved by DARTS is left to future work. The full version shows a join that can be performed in subquadratic time despite its m-width/submodular width being $= 2$. Thus the problem of precisely characterizing which joins can be performed in subquadratic time remains open. Moreover, we focused entirely on using degree information for join processing; using other kinds of information stored by databases to improve join processing is a promising direction for future work.

Acknowledgements. The authors would like to thank Atri Rudra for pointing out the connection to submodular width. CR gratefully acknowledges the support of the Defense Advanced Research Projects Agency (DARPA) XDATA Program under No. FA8750-12-2-0335 and DEFT Program under No. FA8750-13-2-0039, DARPAs MEMEX program under No. FA8750-14-2-0240, the National Science Foundation (NSF) under CAREER Award No. IIS-1353606, Award No. CCF-1356918 and EarthCube Award under No. ACI-1343760, the Office of Naval Research (ONR) under awards No. N000141210041 and No. N000141310129, the Sloan Research Fellowship, the Moore Foundation Data Driven Investigator award, and gifts from American Family Insurance, Google, Lightspeed Ventures, and Toshiba.

References

1 F. Afrati, M. Joglekar, C. Ré, S. Salihoglu, and J. Ullman. GYM: A multiround join algorithm in mapreduce. *CoRR*, abs/1410.4156, 2014. URL: `http://arxiv.org/abs/1410.4156`.

2 F. N. Afrati and J. D. Ullman. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE TKDE*, 23, 2011.

3 N. Alon, I. Newman, A. Shen, G. Tardos, and N. Vereshchagin. Partitioning multidimensional sets in a small number of "uniform" parts. *Eur. J. Comb.*, 28, 2007.

4 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles (extended abstract). In *Proceedings of the Second Annual European Symposium on Algorithms*, ESA'94, pages 354–364, London, UK, 1994. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=647904.739463`.

5 A. Atserias, M. Grohe, and D. Marx. Size Bounds and Query Plans for Relational Joins. *SIAM J. Comput.*, 42, 2013.

6 Ilya Baran, ErikD. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3sum. In F. Dehne, A. Lopez-Ortiz, and J. Sack, editors, *Algorithms and Data Structures*, volume 3608 of *Lecture Notes in Computer Science*, pages 409–421. Springer Berlin Heidelberg, 2005. `doi:10.1007/11534273_36`.

7 P. Beame, P. Koutris, and D. Suciu. Communication Steps for Parallel Query Processing. In *PODS*, 2013.

8 P. Beame, P. Koutris, and D. Suciu. Skew in Parallel Query Processing. In *PODS*, 2014.

9 C. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. *TCS*, 239, 2000.

10 G. Gottlob, M. Grohe, M. Nysret, S. Marko, and F. Scarcello. Hypertree Decompositions: Structure, Algorithms, and Applications. In *WG*, 2005.

11 J. Gross, J. Yellen, and P. Zhang. *Handbook of Graph Theory, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2013.

12 M. Joglekar and C. Ré. It's all a matter of degree: Using degree information to optimize multiway joins. *CoRR*, abs/1508.01239, 2015. URL: `http://arxiv.org/abs/1508.01239`.

13 P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS'11, pages 223–234, New York, NY, USA, 2011. ACM. `doi:10.1145/1989284.1989310`.

14 J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014. URL: `http://snap.stanford.edu/data`.

15 D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60, 2013.

16 H. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms: [extended abstract]. In *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS'12, pages 37–48, New York, NY, USA, 2012. ACM. `doi:10.1145/2213556.2213565`.

17 H. Ngo, C. Ré, and A. Rudra. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *SIGMOD*, 42, 2014.

18 T. Veldhuizen. Leapfrog triejoin: a worst-case optimal join algorithm. *CoRR*, abs/1210.0481, 2012. URL: `http://arxiv.org/abs/1210.0481`.

19 M. Yannakakis. Algorithms for Acyclic Database Schemes. In *VLDB*, 1981.

Filtering With the Crowd: CrowdScreen Revisited

Benoît Groz[1], Ezra Levin[2], Isaac Meilijson[3], and Tova Milo[4]

1 **Université Paris Sud 11, Orsay, France**
benoit.groz@lri.fr
2 **Tel Aviv University, Tel Aviv, Israel**
ezralevin@gmail.com
3 **Tel Aviv University, Tel Aviv, Israel**
isaco@post.tau.ac.il
4 **Tel Aviv University, Tel Aviv, Israel**
milo@cs.tau.ac.il

Abstract

Filtering a set of items, based on a set of properties that can be verified by humans, is a common application of CrowdSourcing. When the workers are error-prone, each item is presented to multiple users, to limit the probability of misclassification. Since the Crowd is a relatively expensive resource, minimizing the number of questions per item may naturally result in big savings. Several algorithms to address this minimization problem have been presented in the CrowdScreen framework by Parameswaran et al. However, those algorithms do not scale well and therefore cannot be used in scenarios where high accuracy is required in spite of high user error rates. The goal of this paper is thus to devise algorithms that can cope with such situations. To achieve this, we provide new theoretical insights to the problem, then use them to develop a new efficient algorithm. We also propose novel optimizations for the algorithms of CrowdScreen that improve their scalability. We complement our theoretical study by an experimental evaluation of the algorithms on a large set of synthetic parameters as well as real-life crowdsourcing scenarios, demonstrating the advantages of our solution.

1998 ACM Subject Classification H.3.3 Information filtering

Keywords and phrases CrowdSourcing, filtering, algorithms, sprt, hypothesis testing.

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.12

1 Introduction

CrowdSourcing for Filtering

Building upon a flourishing ecosystem of CrowdSourcing platforms, a new kind of database systems such as CrowdDB and Qurk endeavors to exploit human inputs to extract or process information [20, 8]. Queries in these systems rely on a small set of basic operators to elicit missing information from the crowd. This triggered a new line of research devoted to the optimization of such basic operations as Joins, Ordering, Aggregates, Selection, etc., in a CrowdSourcing environment [19, 18]. In this paper we focus on the Selection operation, i.e., using the crowd to filter the items satisfying some specific property.

As an example, assume we are sensitive to gluten and would like to know which food items, out of a given list or a menu, may be problematic for us. Scanning food recipes and labels could give information on each individual item, but this is a time consuming job, and the results may be incorrect, e.g. due to some ignored factors such as cross-contamination issues. Asking the Crowd about their knowledge/experience with the product may provide an alternative solution to the problem. However, contributors will sometimes provide erroneous

answers, so that multiple answers must be gathered in order to ascertain that a product is gluten-free. But how many people need to be asked? Let us assume that (1) the probability that each category of food contains the ingredients, and (2) the error rates among the answers (false positives and false negatives rates) are prior knowledge – we will briefly discuss this assumption later. For example, suppose that some category of dishes, e.g. cereal, contains gluten with probability 0.5, and suppose the probability of false positives and false negatives are both 0.4. How can we decide with an average precision of 90% whether or not a given cereal contains some gluten, and how many answers will be required for that?

A simple solution is to fix in advance some budget m for answers, and then decide that our dish contains (resp. does not contain) gluten as soon as we get more than $m/2$ positive (negative) answers. For our parameters, one can easily check that a budget of $m = 41$ answers is required to obtain 90% precision with this strategy. Naturally, we do not always have to use the full budget – as soon as 21 positive (or negative) answers are obtained we can stop and make a decision with the required precision. We will thus ask between 21 and 41 questions and, on average, about 34 questions (we omit the exact computation). Note however that a smaller average number of questions can be used if we employ a more efficient strategy, known in the literature as a *sequential test* [23], and adapt dynamically the budget as answers are received. We can for instance show that on average only 23 answers are sufficient to reach a decision if we use the following strategy which also guarantees an average precision of 90%:

- claim there is gluten as soon as the number of positive answers exceeds the number of negative answers by 6
- claim there is none when negative answers exceed positive answers by 6
- use majority vote in the absence of conclusion after 51 questions

More generally, the challenge that we try to address in this paper is devising tests that minimize the expected number of answers required from the crowd for deciding whether a given object satisfies a selection criteria, while guaranteeing that the average error stays below the required threshold.

The CrowdScreen Framework

The problem of minimizing the number of questions needed to classify items accurately clearly predates CrowdSourcing and we discuss related work in the conclusion. Yet, CrowdSourcing scenarios may be particular in the sense that errors on specific items are typically tolerated as long as a good accuracy is guaranteed on average over the whole set of items [22].

To study the optimization of filtering, we adopt in this paper a simple and general model by Parameswaran et al. [22], whose purpose is to compute optimal querying strategies. They define a (deterministic) strategy as a function mapping the number of positive and negative answers received from the users to a decision in $\{\texttt{Pass}, \texttt{Fail}, \texttt{Cont}\}$. A `Pass` (resp. `Fail`) decision signifies we stop asking questions and accept the object in question as satisfying the filter (resp. reject the object), and `Cont` stands for asking additional questions. They also consider probabilistic strategies that map each point to both a probability to stop asking questions and the decision (`Pass` or `Fail`) in case the strategy terminates at this point. Questions to the Crowd are considered expensive and therefore the maximal number of questions allotted to the strategy is bounded by some fixed budget. The selectivity of the filter and the error rates of the answers, as previously mentioned, are considered prior knowledge in the model. A problem instance thus consists of a budget bound, a maximal bound on the expected error authorized for the strategy, and those prior probabilities.

Several algorithms and heuristics have been introduced in [22] to compute deterministic and probabilistic strategies. While these algorithms are efficient for very small budgets (up to 14 questions per item), larger sample sizes were hardly considered. In fact, the presented algorithms are not a good fit for larger budget as they either have high complexity (exponential, or polynomial but with high degree), or suffer from numeric instability, hence do not always return a strategy meeting the error constraint.

The restriction to small budgets may be justified by the assumption that CrowdSourcing applications typically use little redundancy. Yet 14 questions are not sufficient to filter items with a high precision when the error rates are high: our motivating example for instance requires more than 40 questions, and the original works about sequential tests in statistical testing [23, 2] generally consider budgets featuring hundreds or thousands of answers. *The goal of this paper is thus to devise algorithms that scale well for large budgets.*

Contributions

Our contributions are three-fold. First, we provide new theoretical insights to the problem. We then devise efficient algorithms based on these insights. We also propose optimizations of algorithms in [22] to improve their scalability.

Specifically, we first show, in Section 2, that key properties of the problem derive from well-known results on the *likelihood ratio* (to be formally defined). We exploit these in Section 3 to devise a scalable algorithm: `AdaptSprt` inspired from the popular SPRT [23]. We then revisit, in Section 4, the heuristics from [22]. In particular, we show that their method that enumerates all (ladder-shaped) strategies has complexity $O(2^{2m})$, and we present optimizations extending the range of budgets for which this enumeration is tractable by a factor $\approxeq 1.5$. We similarly show that the `shrink` heuristic from [22], which computes slightly suboptimal deterministic strategies, can be optimized to run in $O(m^4)$ instead of $O(m^5)$, and further establish, in Section 5, connections between deterministic and probabilistic strategies. In particular we show that an optimal probabilistic strategy can be computed through a minor modification to the `shrink` strategy, as an alternative to the linear programming approach that was considered there (and which we show to suffer from numeric instability). For space constraints we defer some of the proofs to the technical report [11]. Finally, to complement our theoretical study we briefly illustrate (with more details in the technical report) the practical advantages and limitations of our solutions by a set of experiments on (1) a large set of synthetic parameters and (2) a small real-life scenario.

2 Preliminaries

We first list definitions and notations, as well as general properties we use to devise efficient strategies. The formal introduction below follows [22] and we diverge afterwards.

2.1 Definitions

We wish to harness the wisdom of the crowd to determine, for each object O of a large dataset D, whether the object has property V ($V = 1$) or not ($V = 0$). We thus ask users in the crowd if they believe the object has the property. To compensate for possible mistakes, we query multiple users until we have gathered enough evidence to reach a pass/fail decision about O. The selectivity ratio s (percentage of objects in D having property V) and the users' error rates are assumed prior knowledge. We thus define the error rates e_0 and $e_1 < 0.5$ as the probability of a user error, given that $V = 0$ and $V = 1$ respectively. We briefly discuss

in the conclusion how these values can be estimated. Finally, we consider that each question has a unit cost, and specify a *budget constraint* m; the maximal number of questions we are allowed to ask before reaching a decision on item O.

Strategies

The sequence of answers received when classifying an item can be visualized as a walk on a discrete 2-dimensional grid where the x and y axes represent the number of negative and positive answers received. The current state of the sequence is the point (x, y) matching the number of positive and negative answers received. The transitions between states match the answers provided by the users: if a negative answer is received in state (x, y), the state moves to $(x + 1, y)$. If a positive answer is received instead, the state moves to $(x, y + 1)$. For each point on the grid we define $P_{stop}(x, y)$ as the probability that the walk terminates upon reaching point (x, y). When terminating, a claim on the value of V is returned: `Pass` ($V = 1$) or `Fail` ($V = 0$).

A strategy is defined by the function $P_{stop}(x, y)$ mapping each point to the probability of terminating when reaching point (x, y). In a probabilistic strategy $P_{stop}(x, y)$ is taken over the interval $[0, 1]$, whereas in a deterministic strategy $P_{stop}(x, y)$ must be either 0 or 1. Point (x, y) is a *continuing point* if $P_{stop}(x, y) = 0$, a *terminating point* if $P_{stop}(x, y) = 1$, and a *probabilistic* point otherwise. An optimal choice betweeb `Pass` or `Fail` in case we stop can easily be computed from x, y and the input parameters, using a well-known property of the likelihood ratio recalled in Section 2.2 (the choice does not depend on the strategy). A point in which the decision is `Pass` is an *accepting point* and a point in which the decision is `Fail` is a *rejecting point*. The *cost* of a given strategy is the expected number of answers needed in order to reach a decision, while the *error* of the strategy is the probability that the strategy reaches a wrong decision. We formalize this next.

Strategy and Grid characteristics

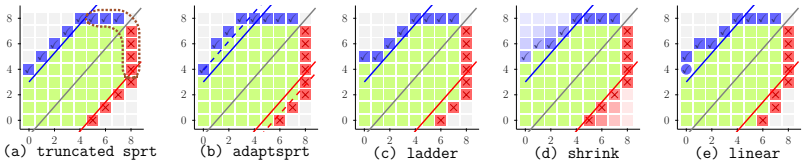
We compute the cost and error of a strategy as mentioned in [22]. Intuitively, our equations first count paths leading to (x, y) according to the strategy, then multiply the result ($Path$) by the probability (S_0, S_1) that answers follow any single such path. Let $S_i(x, y)$ be the probability that by the time we have asked $x + y$ queries we receive (in any specific order) x negative answers and y positive answers and have $V = i$. We then have:

$$S_0(x, y) = (1 - s) \times (1 - e_0)^x \times e_0^y \tag{1}$$

$$S_1(x, y) = s \times e_1^x \times (1 - e_1)^y \tag{2}$$

Let $Path(x, y)$ denote the weighted number of paths (i.e., sequences of answers) that consist of y positive and x negative answers, each path being weighted by $(1-p)$ where p is the probability (depending on the path and the strategy) to stop along the path before reaching point (x, y). We partition these paths into two groups: $Path(x, y) = tPath(x, y) + cPath(x, y)$ with $tPath(x, y) = P_{stop}(x, y) \times Path(x, y)$. Thus, $tPath(x, y)$ and $cPath(x, y)$ respectively count the paths that terminate and continue after reaching point (x, y). We observe that the strategy P_{stop} uniquely determines the values of $tPath$ and $cPath$, and reciprocally. A point (x, y) is *reachable* if $Path(x, y) > 0$.

▶ **Example 1.** The running example illustrates the definitions along the paper with error rates $e_0 = .25$ and $e_1 = .2$, threshold $\tau = .0075$, budget $m = 15$, and selectivity $s = .8$. Figure 1 pictures the strategies returned for those parameters by the algorithms investigated



algorithm	cost	error
SPRT	6.94	0.008
AdaptSprt	7.748	0.00741
ladder	7.59	0.00749
shrink	7.73	0.00748
linear	7.56	0.0075

Figure 1 Strategies returned for $e_0 = .25$, $e_1 = .2$, $\tau = .0075$, $m = 15$, and $s = .8$.

in this paper: unreachable, accepting, rejecting, and continuing points are represented as white, blue (with a checkmark), red (with cross), and green squares respectively. Probabilistic points are circles, with similar colors (and marks). Other signs in the figure will be discussed later on.

We further define $g_i(x, y)$ as the probability that $V = i$ and the point (x, y) is ever reached for $i = 0, 1$. This value is computed as: $g_i(x, y) = Path(x, y) \times S_i(x, y)$. Let $\mathrm{Err}(x, y)$ denote the probability of error when making a decision at point (x, y) (we detail in the next section how to calculate $\mathrm{Err}(x, y)$). The probability that we reach (x, y) and stop there is $\sum_{(x,y)}(g_0(x, y) + g_1(x, y)) \times P_{stop}(x, y)$. The cost of a strategy is therefore:

$$C = \sum_{(x,y)} (g_0(x, y) + g_1(x, y)) \times P_{stop}(x, y) \times (x + y)$$

and the error of the strategy is:

$$E = \sum_{(x,y)} (g_0(x, y) + g_1(x, y)) \times P_{stop}(x, y) \times \mathrm{Err}(x, y)$$

The Problem Definition

The error threshold τ fixes the maximal error a strategy is allowed. A strategy is *feasible* if it satisfies the budget and error constraints m and τ, and *optimal* if it has minimal cost among feasible strategies. Our objective is to find the optimal strategy, given the priors e_0, e_1, s and the constraints m and τ.

Optimal Stopping Problem

Input: selectivity s, error threshold τ, error rates e_0, e_1 and budget m
Question: find a feasible strategy that minimizes C

The strategies (and grids) that we consider satisfy certain constraints, enumerated below. The objective is to minimize the cost C under the following constraints:

1. There is exactly one path going through the origin : $cPath(0, 0) + tPath(0, 0) = 1$
2. Conservation of paths: the weighted number of paths reaching point (x, y) is equal to the number of paths that continue through its predecessors $(x - 1, y)$ and $(x, y - 1)$: $Path(x, y) = cPath(x - 1, y) + cPath(x, y - 1)$
3. All strategies are limited to m queries: $\forall(x, y), x + y = m \implies cPath(x, y) = 0$
4. The error rate of the strategy is at most τ:

$$E = \sum_{(x,y):x+y\leq m} tPath(x, y) \times \min(S_0(x, y), S_1(x, y)) \leq \tau$$

Up till now, the introduction followed the definitions and equations of [22], but the remainder of this section presents useful properties of strategies from a different perspective.

2.2 General Framework

The probability that $V = i$ given that (x, y) has been reached is given by: $g_i(x, y)/(g_0(x, y) + g_1(x, y)) = 1/(1 + (g_{(1-i)}(x, y)/g_i(x, y)))$, where $g_i(x, y)$, as previously defined, is the probability that $V = i$ and point (x,y) is reached for $i = 0, 1$. The error committed when making a decision at (x, y) is therefore:

$$\mathrm{Err}(x, y) = \begin{cases} \frac{1}{1+g_1(x,y)/g_0(x,y)} & \text{if the decision at } (x, y) \text{ is } \texttt{Pass} \\ \frac{1}{1+g_0(x,y)/g_1(x,y)} & \text{if the decision at } (x, y) \text{ is } \texttt{Fail} \end{cases} \tag{3}$$

To minimize error, a strategy should therefore opt for `Pass` if $g_1(x, y)/g_0(x, y) > 1$, and `Fail` if $g_1(x, y)/g_0(x, y) < 1$. The decision has no impact on error when $g_1(x, y) = g_0(x, y)$. We henceforth assume that all strategies adopt this decision rule since it minimizes error and has no impact on the cost. The decision to accept or reject thus only depends on the value of the *likelihood ratio* $g_1(x, y)/g_0(x, y)$, which can be computed from x, y, and the parameters independently from the strategy. The following equation further details the location of accepting and rejecting points, and as such refines the property presented as the *path principle* in [22]:

$$\begin{aligned} \log \frac{g_1(x, y)}{g_0(x, y)} &= \log \left(\frac{s}{1-s} \times \left(\frac{e_1}{1-e_0} \right)^x \times \left(\frac{1-e_1}{e_0} \right)^y \right) \\ &= \log \frac{s}{1-s} + x \log \left(\frac{e_1}{1-e_0} \right) + y \log \left(\frac{1-e_1}{e_0} \right) \end{aligned}$$

▶ Remark. The contour lines for the likelihood ratio (i.e., the set of points with likelihood ratio $g_1(x, y)/g_0(x, y) = c$ for some constant c) form a straight line on the grid, and all contour lines are parallel. Furthermore $e_0, e_1 < 1/2$ so the ratio increases strictly with y and decreases with x.

We call the line $(g_1(x, y)/g_0(x, y)) = 1$ the *decision line*. Points above this line satisfy $1 < g_1(x, y)/g_0(x, y)$ and are therefore accepting, while points below the line are rejecting.

▶ **Example 2.** For the running example with $e_0 = .25$, $e_1 = .2$, $\tau = .0075$, $m = 15$ and $s = .8$, the decision line has equation: $y = -\frac{\log(.2/.75)}{\log(.8/.25)}x - \frac{\log(4)}{\log(.8/.25)} \approxeq 1.11 \times x - 2$. This line is depicted in grey on all the grids in Figure 1.

2.3 Simple Optimizations

Before presenting the algorithms we describe three basic optimizations that they all employ. The first is borrowed from [22] and the other two are new.

Ladder Strategies

Parameswaran et al. [22] prove that under reasonable assumptions, all optimal strategies have a particular shape. They define a *ladder strategy* as a strategy whose terminating points can be partitioned into two converging sequences: the *upper ladder* and the *lower ladder*. The points of an upper ladder are given by a non-decreasing mapping from x to y whereas the lower ladder is a non-decreasing mapping from y to x. Furthermore, the points of the upper ladder stay above the decision line, whereas those of the lower ladder stay below. For example, all deterministic strategies represented in Figure 1 (i.e., a,b,c, and d) are ladder strategies. It has been conjectured in [22] that any optimal strategy is a ladder strategy. We adopt this conjecture and focus in this paper on ladder strategies.

Pruning the Grid

Let $(x_{\texttt{dec}}, y_{\texttt{dec}})$ denote the point at which the decision line and $x+y=m+1$ intersect, i.e., the unique point such that $x+y=m+1$, $(x-1,y)$ is accepting and $(x,y-1)$ rejecting. All points with $x=x_{\texttt{dec}}$ or $y=y_{\texttt{dec}}$ are terminating in any optimal strategy, since all points reachable from (x,y) return the same decision (e.g., `Pass`) so that continuing asking questions from (x,y) is pointless.

▶ **Example 3.** In the running example, the budget bounds $x+y$ by 15, so that $(x_{\texttt{dec}}, y_{\texttt{dec}}) = (8,8)$. All strategies presented in Figure 1 are therefore restricted to $x, y \leq 8$.

Deciding Feasibility

A problem instance admits a feasible strategy (strategy meeting the error and budget constraints) if and only if the rectangular strategy $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$ with terminating points only along $x=x_{\texttt{dec}}$ and $y=y_{\texttt{dec}}$ is feasible. Point $(x_{\texttt{dec}}, y_{\texttt{dec}})$ is obtained in constant time as the intersection of two lines: the decision line and the line $x+y=m$. The error of $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$ can thus be computed as $B(e_0; y_{\texttt{dec}}, x_{\texttt{dec}}) + B(e_1; x_{\texttt{dec}}, y_{\texttt{dec}})$, where B denotes the incomplete beta function [7], incorporated in standard numeric libraries. One can thus decide feasibility in constant time for all practical purposes, and we therefore only consider feasible problems from now on.

3 Likelihood Ratio Test

The first solution we introduce is based on the Sequential Probability Ratio Test (SPRT), defined by Wald [23] in the context of quality control. As it may return strategies with unbounded budgets, we also consider its truncated variant which limits the budget but may exceed the error constraint. We finally propose an adapted version of SPRT to accommodate both budget and error constraints.

3.1 SPRT: Definition and Boundaries

General SPRT: Infinite Budget

The SPRT strategy defined by Wald [23] is the strategy that continues asking questions until the likelihood ratio (defined in Section 2.2) leaves interval $]\alpha, \beta[$, where α and β depend on the error we are willing to tolerate under $V=0$ and $V=1$. To continue asking questions until reaching a point with $\mathrm{Err}(x,y) \leq \tau$, we thus set $\alpha = \frac{\tau}{1-\tau}$, $\beta = \frac{1-\tau}{\tau}$. The error in each decision point (hence the overall error of the strategy) is bounded by τ. As a corollary of Remark 2.2, grid points where $\mathrm{Err}(x,y) > \tau$ are bound by two parallel lines, so the continuing points of the SPRT strategy are the points located between these SPRT lines, characterized in the following Proposition:

▶ **Proposition 4.** *A point (x,y) satisfies $Err(x,y) \leq \tau$ if and only if $g_1(x,y)/g_0(x,y) \notin]\frac{1-\tau}{\tau}, \frac{\tau}{1-\tau}[$. Furthermore, the points with $Err(x,y) \leq \tau$ are either the points above the line: $\log\left(\frac{1-\tau}{\tau} \times \frac{1-s}{s}\right) \leq x \log\left(\frac{e_1}{1-e_0}\right) + y \log\left(\frac{1-e_1}{e_0}\right)$ (accepting points) or the points below the line: $\log\left(\frac{\tau}{1-\tau} \times \frac{1-s}{s}\right) \geq x \log\left(\frac{e_1}{1-e_0}\right) + y \log\left(\frac{1-e_1}{e_0}\right)$ (rejecting points). We note that both lines have identical slopes and are therefore parallel.*

▶ **Example 5.** In the running example, the equations of the SPRT lines are approximately $1.11 \times x - 2 \pm 4.2$. To facilitate the comparison of strategies, the SPRT lines are represented in blue and red on every plot of Figure 1.

As shown by Wald [23], this property allows to approximate in constant time the expected cost of the SPRT. Even though an arbitrary number of questions may be needed to reach a decision, the expected cost is typically small [23].

Limitations

Although SPRT is optimal when the budget for questions is unlimited [23], it yields unbounded strategies which may issue an arbitrary (possibly infinite) number of questions, and is thereby not suitable for our limited budget.

Truncated SPRT

To limit the maximum number of questions, Wald also introduces the truncated SPRT, similar to SPRT, except that all points with $x + y = m$ are terminating to guarantee a decision is reached after at most m questions. Obviously, we then prune the strategy along the lines $y = y_{\texttt{dec}}$ and $x = x_{\texttt{dec}}$ as detailed in Section 2.3.

▶ **Example 6.** Figure 1(a) represents the truncated SPRT strategy for the running example with brown dots around the truncation points. Its error; 0.008, exceeds τ, because the truncation includes some decision points with $\mathrm{Err}(x, y) > \tau$. To compensate for this additional error, any feasible strategy must therefore include points further from the SPRT lines.

The truncated SPRT provides a strategy within constant time, since one only needs to compute the likelihood ratio r of the current point (x, y) to decide whether to continue $r \in]\frac{\tau}{1-\tau}, \frac{1-\tau}{\tau}[$, accept $(r \geq \frac{1-\tau}{\tau})$ or reject $(r \leq \frac{\tau}{1-\tau}])$.

But the error of the strategy may be larger than τ since the truncation points have error larger than τ. In some instances, the truncated SPRT still returns a feasible strategy, e.g. when some decision points along the SPRT lines have an error slightly less than τ, thus compensating for the additional error caused by the truncation. But feasibility is not always guaranteed, and therefore the truncated SPRT cannot be trusted to solve our problem.

3.2 Adapting the SPRT Threshold

As SPRT cannot be trusted to provide feasible strategies, we propose a new adaptation of the SPRT strategy, called `AdaptSprt`, which preserves the simplicity of the SPRT approach but always returns a feasible solution.

Intuitively, the `AdaptSprt` algorithm computes the best strategy whose terminating points form two lines, parallel and equidistant to the decision line, plus truncation points along $x = x_{\texttt{dec}}$ and $y = y_{\texttt{dec}}$. In other words, `AdaptSprt` starts from initial strategy $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$ and turns the points further from the decision line into terminating points, as long as the error of the strategy remains below the authorized threshold. This guarantees that a feasible strategy will always be returned when there is one. For efficiency, we use binary search to determine which points can be turned into terminating points.

Algorithm `AdaptSprt` can be defined more formally in terms of the likelihood ratio. For all $\eta > 0$ let σ_η be the strategy that continues asking questions on all points (x, y), $x \leq x_{\texttt{dec}}, y \leq y_{\texttt{dec}}$ where the likelihood ratio belongs to $]1/\eta, \eta[$. `AdaptSprt` computes the maximal threshold η for which σ_η is feasible. Algorithm 1 details the steps in `AdaptSprt`. We first build in $O(m^2 \log m)$ a list of all points (x, y) with $x < x_{\texttt{dec}}$ and $y < y_{\texttt{dec}}$, ordered by increasing likelihood ratio r (lines 1,2 of Algorithm 1). The continuing points of the `AdaptSprt` strategy will be the first i points from the list, for some index i. As the error (resp. the cost) increases

Algorithm 1: `AdaptSprt` (e_0, e_1, τ, m, s)

```
1  for all x, y, compute r(x, y) = g1(x, y)/g0(x, y)
2  L ← points (x, y) ordered by increasing r(x, y)
3  Compute i0 = min{i | EvalErr(i, L) < τ}
4  return r(L(i0))

   procedure EvalErr(i : int, L : point list)
5      for j in {0, ..., i − 1}
6          P_stop(L(j)) ← 0
7      for j in {i + 1, ...}
8          P_stop(L(j)) ← 1
9      Compute and return the error of strategy P_stop
```

(resp. decreases) with i, the optimal strategy of this form is obtained by computing the minimal i that gives a feasible strategy. The strategy corresponding to index i is evaluated by procedure `EvalErr` in $O(m^2)$, and we can use binary search to compute the minimal index within $\log(m^2)$ iterations. Hence an overall complexity of $O(m^2 \log(m))$.

To represent the `AdaptSprt` strategy, the value r of the likelihood ratio of the i^{th} point is sufficient: when asking queries we can calculate in constant time whether a point has likelihood ratio between $1/r$ and r, and thus reconstruct the grid on the fly. We can also compute the strategy P_{stop} from the list and the index i, as shown in procedure `EvalErr` from Algorithm 1 if a grid representation is preferred.

▶ **Proposition 7.** *The (time) complexity of* `AdaptSprt` *is* $O(m^2 \log(m))$.

▶ **Example 8.** Figure 1(b) presents the `AdaptSprt` strategy for the running example, with termination lines represented as dashed lines. The truncation of the SPRT raises the error substantially above τ, so that `AdaptSprt` must adopt a likelihood ratio threshold η much larger than $(1-\tau)/\tau$ to compensate for the truncation. The dashed lines are thus almost one question beyond those of SPRT.

4 Deterministic Algorithms

We next investigate the scalability of algorithms proposed by Parameswaran et al. [22] for computing strategies. Specifically, we analyze the complexity of these algorithms and present optimizations that drastically reduce the running time of the algorithms compared to the more naïve versions presented in [22], thereby allowing to support larger budgets.

4.1 Enumeration of Ladder Strategies

The most naïve approach to compute an optimal strategy is to enumerate and evaluate all possible strategies. This naïve approach has complexity $O(m^2 \times 2^{m^2/2})$ and is thus intractable.[1] Parameswaran et al. [22] therefore proposed the `ladder` algorithm which limits the search to ladder strategies, as explained in Section 2.3. They report running times that are reasonable for small values of m ($m \leq 14$), but grow exponentially and become unfeasible

[1] A bound of $m^2 \times 2^m$ was improperly claimed in [22] for the naïve enumeration of grids but it is clear from their proof that the actual bound is $O(m^2 \times 2^{m^2/2})$[21]

as m gets larger. We first establish a tight exponential bound for the complexity of `ladder`, and then introduce optimizations offering much shorter running time in practice, in spite of a similar worse case exponential complexity.

Asymptotic Analysis

We prove that the complexity of ladder is essentially $O(2^{2m})$. Our exponential bound bears witness to the efficiency of the `ladder` algorithm relative to the enumeration of all possible (not necessarily ladder-shaped) strategies. For this we can easily show the following lower bound:

▶ **Lemma 9.** *The number of possible upper and lower ladders can be roughly bounded by $O(2^m/\sqrt{m})$. Hence, there are $O(2^{2m}/m)$ deterministic ladder strategies.*

This is an overapproximation, yet a fairly accurate one: we show in the technical report [11] that for $s = 0.5, e_0 = e_1$, the number of ladder strategies is $\Omega(2^{2m}/m^3)$.

Enumeration with Incremental Evaluation

We detail in Algorithm 2 an optimized implementation that computes the cost and error of every ladder strategy incrementally, in overall $O(2^{2m})$. We first discuss our representation of a ladder strategy and then explain our optimizations.

As mentioned in Section 2.3, a ladder strategy consists of two distinct sequences of points: the upper ladder and the lower ladder. Each ladder is represented as an array with size $x_{\texttt{dec}}$ storing integers from -1 up to $y_{\texttt{dec}}$. Array `up` and `down` represent respectively the upper and lower ladder: $\texttt{down}(i)$ and $\texttt{up}(i)$ record respectively the lowest and highest reachable points on column i according to the strategy.

▶ **Example 10.** Figure 1(c) represents the optimal ladder strategy for our input parameters: $\texttt{up} = [5, 5, 6, 7, 8, 8, 8, 8]$ and $\texttt{down} = [-1, \ldots, -1, 0, 1]$. None of the other algorithms depicted returns the optimal strategy on that instance, although the performances are quite similar.

We adapt an old technique (see [14, Algorithm P]) to iterate over all upper ladders in increasing lexicographic order, and enumerate for each one the lower ladders in decreasing order. As a result, arrays representing successive strategies generally differ on the last few columns only, which reduces the amount of work required to evaluate a strategy.

Two simple optimizations allow us to speedup the enumeration: (1) we evaluate incrementally the cost and error of strategies, and (2) we skip some strategies that cannot contribute an optimal solution. For this, we store two arrays `errorTill` and `costTill`, where $\texttt{errorTill}(i)$ records the partial sum of E restricted to the points with $x \leq i$, and similarly with `costTill` for C. We update `errorTill`, `costTill`, and *Path* from one strategy to the next (line 7 of Algorithm 2). The iterator $\texttt{down}.next()$ returns $(-1, [])$ if `down` is already the minimal ladder, and otherwise returns the greatest possible ladder $\texttt{down}'$ smaller than `down`, together with the smallest index i in which `down` and $\texttt{down}'$ differ. To skip hopeless candidates, we set $\texttt{down}(j)$ to $\texttt{down}(i)$ for all $j > i$ when the error up to column i exceeds the threshold, or the cost up to column i exceeds the cost of the best strategy encountered so far (line 13 in Algorithm 2).

▶ **Example 11.** When experimenting on the running example, more than half the strategies were skipped in line 13, and the average index i was 5.5. Some 16 points were visited per strategy, on average, when updating arrays and matrix in line 7, instead of $\approxeq 56$ without incremental evaluation.

Algorithm 2: `ladder` (e_0, e_1, τ, m, s)

```
1  errorTill, costTill ← [0, 0, ..., 0]
2  BestCost ← m + 1
3  BestStrategy ← Null
4  for up in upperladders
5      down ← maximal lowerladder; i ← 0
6      while i ≥ 0
7          Update errorTill, costTill, Path
8          if (errorTill[m] < τ and
9                  costTill[m] < BestCost[m] )
10             BestCost ← costTill[m]
11             BestStrategy ← (up, down)
12         if (errorTill[i] > τ)
13             skip ladders until down(i) is modified
14         else (i, down) ← down.next()
15 return BestStrategy
```

We show in the technical report [11] that the average number of cells updated on line 7 is m. As a consequence, Algorithm 2 has complexity $O(2^{2m}/m) \times O(m)$.

▶ **Proposition 12.** *Algorithm 2 runs in* $O(2^{2m})$.

We have thus proved that an optimal ladder can be obtained in $O(2^{2m})$, and the number of possible ladders strategies is exponential. This does not preclude the existence of faster algorithms, and we leave lower bounds on the complexity of the problem for future research.

4.2 Shrink

Another interesting heuristic-based algorithm introduced by Parameswaran et al. [22] is `shrink`. The strategies returned by this heuristic are not necessarily optimal, but are hardly worse than the optimal ladder strategy in practice, while the running time is much improved. A naïve implementation following [22] has complexity $O(m^5)$ and therefore, does not scale well for large values of m. We next show how `shrink` can be run in $O(m^4)$.

We recall the `shrink` heuristic from [22] in Algorithm 3. This algorithm starts with the initial strategy $\sigma_{\texttt{triangle}}(m)$ having terminating points along the line $x + y = m$. At each iteration, for each terminating point (x, y) on the grid, we check if the solution would remain feasible if we were to turn one of the neighboring points $(x - 1, y)$, $(x, y - 1)$ into a terminating point. For all feasible point we calculate the change in cost ΔC and error ΔE that would result from shrinking the point. We then shrink the point with the largest ratio $-\frac{\Delta C}{\Delta E}$ and repeat this step until no more points can be shrunk. We thus use ratio $-\frac{\Delta C}{\Delta E}$ in order to maximize the cost removed from the strategy while minimizing the additional error.

▶ **Example 13.** In Figure 1(d), we shade points that were turned into terminating points along the successive iterations of `shrink`, with darker points corresponding to later iterations[2]. The first point is thus $(0, 7)$, followed by $(1, 7), (0, 6), \ldots, (6, 1)$, and $(5, 0)$.

[2] Terminating points with $x = 8$ or $y = 8$ are particular in that they were not shrinked but were terminating from the beginning. We color them in dark red and blue.

Algorithm 3: `shrink` (e_0, e_1, τ, m, s)

Compute $S_0, S_1, x_{\texttt{dec}}, y_{\texttt{dec}}$
for all x, y:
$$P_{stop}(x,y) \leftarrow \begin{cases} 1 \text{ if } x+y=m, \\ 0 \text{ otherwise} \end{cases}$$
Compute $Path$, Δ_{Cost} and Δ_{Err}
$\texttt{Error} \leftarrow \Delta_{\text{Err}}(0,0)$
$S \leftarrow \{$(x,y) along the boundary $|$ $\texttt{Error} + Path(x,y) \times \Delta_{\text{Err}}(x,y) < \tau\}$
while $S \neq \emptyset$
  $(x_0, y_0) \leftarrow$ point of S maximizing $-\frac{\Delta_{\text{Cost}}(x,y)}{\Delta_{\text{Err}}(x,y)}$
  $P_{stop}(x_0, y_0) \leftarrow 1$
  for all x, y: update $Path$, Δ_{Cost}, Δ_{Err}
  update S
return P_{stop}

Algorithm `shrink` from [22] is polynomial, but still pretty slow. We next present new equations for the ratios together with a pruning optimization, that make it run faster.

4.2.1 Computing Cost/Error Ratios Efficiently

A major source of inefficiency in the above `shrink` implementation is the calculation of the Cost/Error ratio in each iteration. The naïve implementation of `shrink` computes the Cost/Error ratio separately for each terminating point on the grid, by evaluating the cost and error of the shrunken strategy. As there are $\Omega(m)$ terminating points this requires $\Omega(m^3)$ operations per iteration. We introduce new equations that help compute $\Delta_{\text{Cost}}(x,y)$ and $\Delta_{\text{Err}}(x,y)$ for all points (x,y), with overall complexity $O(m^2)$. Algorithm `shrink` was initially designed to compute deterministic strategies, but in Section 5 we extend it to probabilistic strategies, so we present all equations in a general probabilistic setting.

Impact of Modifying the Probability to Stop

Let us denote by $\texttt{CostImpact}(x,y)$ and $\texttt{ErrorImpact}(x,y)$ the average contributions to cost and error of one single path through (x,y) (and possibly stopping at (x,y)). We then have:

$$\texttt{CostImpact}(x,y) = P_{stop}(x,y) * X + (1 - P_{stop}(x,y)) * Y$$
$$\texttt{ErrorImpact}(x,y) = P_{stop}(x,y) * Z + (1 - P_{stop}(x,y)) * T$$

where X, Y, Z and T are defined as

$$X = (S_0(x,y) + S_1(x,y)) * (x+y) \qquad Y = \texttt{CostImpact}(x+1,y) + \texttt{CostImpact}(x,y+1)$$
$$Z = \min(S_0(x,y), S_1(x,y)) \qquad T = \texttt{ErrorImpact}(x+1,y) + \texttt{ErrorImpact}(x,y+1)$$

Intuitively, X and Z are the contribution to the overall cost and error from any sequence of x negative answers and y positive answers, whereas Y and T are inductively defined as the contribution to cost and error of a path traversing the node. To compute the impact of modifying the strategy at (x,y) in terms of these expressions, let E, E' and C, C' denote the cost and error of the strategy before and after adding $\delta \in [-1,1]$ to $P_{stop}(x,y)$. Then

$E' - E = \delta \times Path(x,y) \times \Delta_{\text{Err}}$ and $C' - C = \delta \times Path(x,y) \times \Delta_{\text{Cost}}$ where

$$\Delta_{\text{Cost}} = X - Y \qquad \text{and} \qquad \Delta_{\text{Err}} = Z - T \tag{4}$$

We observe in these equations that the Cost/Error ratio is independent of δ and $Path(x,y)$, and is given by $\gamma(x,y) = (T-Z)/(X-Y)$. `CostImpact` and `ErrorImpact` can be computed recursively in $O(m^2)$ over the whole grid, starting from point $(x_{\texttt{dec}}, y_{\texttt{dec}})$. We have thus proved that Δ_{Err} and Δ_{Cost} can be computed at all points in overall $O(m^2)$ according to Equations 4. Furthermore, *Path* can also be computed in $O(m^2)$ according to the preliminaries, so that each iteration of `shrink` takes time $O(m^2)$. In addition, there are at most $O(m^2)$ such iterations, since the number of iterations is at most the number of squares on the grid. Therefore, our implementation of shrink runs in $O(m^4)$.

▶ **Proposition 14.** *With our optimizations, the* `shrink` *algorithm runs in* $O(m^4)$.

Note however that the actual number of iterations is proportional to the number of points removed from the grid so the running time is quadratic when few points are removed.

4.2.2 Minimizing Shrink Iterations

To further speed up the computation we show how the pruning optimization described in subsection 2.3 can spare about half the iterations. Specifically, we prune the initial strategy $\sigma_{\texttt{triangle}}(m)$ into $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$. To justify this move, we show in the technical report [11] that the points that are pruned are anyway the first points eliminated by `shrink`.

▶ **Proposition 15.** *The first iterations of* `shrink` *from the initial strategy* $\sigma_{triangle}(m)$ *eliminate the points with* $x > x_{dec}$ *or* $y > y_{dec}$ *until the strategy* $\sigma_{rect}(x_{dec}, y_{dec})$ *is considered. Consequently the solutions returned by* `shrink` *from initial strategy* $\sigma_{triangle}(m)$ *and from* $\sigma_{rect}(x_{dec}, y_{dec})$ *are identical.*

▶ Remark. Another heuristic, symmetric to `shrink`, was introduced in [22]. This `growth` heuristic starts with the initial strategy asking 0 questions: all points are initially terminating, and then iteratively turns terminating points into continuing points. Heuristic `growth` did not always return a feasible strategy, but we show in the technical report [11] that adopting a better initial strategy wipes off the problem. The performances of `growth` and `shrink` are fairly similar, so we do not detail the heuristic further in this paper.

5 Randomized strategies

Previous sections focus on deterministic strategies, for which we have no optimal scalable algorithm. But if we search instead for probabilistic strategies, our optimization problem becomes continuous, and the constraints presented in Section 2.1 are all linear. We can thus use linear programming to compute an optimal solution in PTIME [22]. How are the probabilistic and deterministic strategies related? In particular, can we compute reasonably good deterministic strategies from probabilistic ones?

In this section we first prove that an optimal probabilistic strategy has essentially a *single* probabilistic point (point where P_{stop} differs from 0 and 1). Continuing at this point thus provides a deterministic strategy. Conversely, we show that a minor modification to the `shrink` algorithm allows to compute an optimal probabilistic strategy.

5.1 Randomization is Limited

We prove in the technical report [11] that in any optimal strategy the Cost/Error ratio is the same in all probabilistic points, and this ratio is not greater than the ratio of any terminating point nor smaller than the ratio of any continuing point. We also prove that the probability of terminating can be transferred from a point to any point with higher ratio without increasing error and cost, and exploit this property to prove the following result:

▶ **Proposition 16.** *There exists an optimal probabilistic strategy with a single probabilistic point. Furthermore, in any optimal strategy the probabilistic points maximize the ratio γ among non-terminating points.*

By turning the unique probabilistic point of such a strategy into a continuing point, one thus obtains a deterministic strategy with error less than τ and with slightly larger cost.

▶ **Example 17.** Figure 1(e) represents an optimal probabilistic strategy, with a single probabilistic point, at $(0, 4)$, where the probability of terminating is $P_{stop}(0, 4) \approxeq 0.623$. If we set $P_{stop}(0, 4)$ to 0, the cost rises to $\approxeq 7.789$.

The linear programming techniques mentioned above are very efficient for small values of m, and have polynomial complexity in theory. In practice, however, our experiments with common linear solvers show that they may be rather slow or inaccurate, returning poor strategies even for moderately large budgets. We therefore propose an alternative efficient algorithm based on `shrink` to compute optimal probabilistic strategies.

5.2 Shrink for Randomized Strategies

Algorithm `shrink` as defined in [22] returns a deterministic, not necessarily optimal, strategy, but it can easily be adapted to compute an optimal randomized strategy by replacing lines 5, 6, and 8 with respectively:

- line 5: $S \leftarrow \{(x, y) \mid$ (x,y) is reachable$\}$
- line 6: **while** $S \neq \emptyset$ and $\texttt{Error} < \tau$
- line 8: $P_{stop}(x_0, y_0) \leftarrow \min(1, \frac{\tau - \texttt{Error}}{Path(x,y) \times \Delta_{\text{Err}}(x,y)})$

This new algorithm `shrinkp` still computes the point with the maximal ratio, but adapts the probability of terminating at this point so as not to exceed error τ, instead of restricting the maximum to points on which one can terminate without exceeding error τ. It turns out that `shrinkp` returns an optimal strategy (we leave the proof for the technical report [11]):

▶ **Proposition 18.** *The probabilistic strategy returned by* `shrinkp` *is optimal, and has a single probabilistic point.*

This result sheds a new light on the `shrink` algorithm, but it can also be used to leverage the running time of `shrink` and the linear program, since `shrink` and `shrinkp` coincide at any step until the last iteration of `shrink` as we discuss in the technical report [11].

6 Experimental evaluation

Synthetic and real-crowd experiments: quality of the strategies

To complement the theoretical study we conducted experiments on a large set of synthetic parameters. Due to space constraints we only present a small sample of our experiments here and leave details for the technical report [11]. Those experiments show that some linear

	question	s	e_0	e_1
Q1	photos from Australia	.18	.25	.36
Q2	photos from Greece or Cyprus	.26	.27	.32
Q3	dishes containing dairy	.17	.11	.27
Q4	dishes containing onions	.54	.38	.27
~~Q5~~	~~dishes containing garlic~~	~~.62~~	~~.44~~	~~.48~~
~~Q6~~	~~dishes containing eggs~~	~~.19~~	~~.22~~	~~.57~~

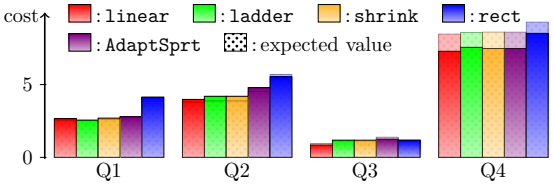


Figure 2 Question parameters(left) and average cost per item (right, with $m = 12$, $\tau = .1$).

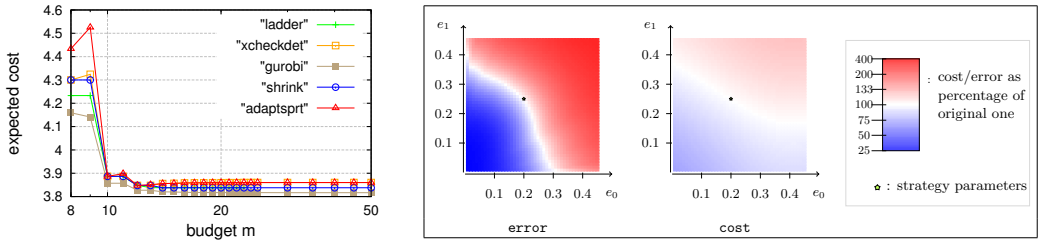


Figure 3 For $e_0 = .2, e_1 = .25, \tau = .05, s = .6$: cost, and sensitivity (only for `shrink` with $m = 15$).

program solvers become unreliable for budgets beyond $m = 30$ questions, while `ladder` times out around $m = 20$ and `shrink` and `AdaptSprt` manage hundreds of questions. The expected cost of strategies matches theoretical expectations with `AdaptSprt` slightly worse than `shrink` and `ladder`, themselves a bit more expensive than the optimal probabilistic strategies. The experiments on a real crowd with budgets up to $m = 40$ exhibit similar patterns. Figure 2 depicts the quality of strategies obtained when asking the crowd to detect (a) the presence of an ingredient in some recipe or (b) the location of a photograph. Experiments were run with a pool of 100 workers on the AskIt [4] crowdsourcing game platform, developed in our lab.

The error rates, summarized in Figure 2, are relatively high because answers were rarely obvious. For question 6 in particular, e_1 was above .5 which means the users more often than not missed the presence of eggs in the dishes. We focus our analysis on questions with reasonable error rates ($Q1$ to $Q4$).

Sensitivity of the model

Applying our algorithms on a real crowd raised new issues such as the adequacy of the model considered. Our algorithms indeed assume the crowd behaves as a random oracle according to error parameters known beforehand. Our synthetic experiment in Figure 3 measures the sensitivity of a strategy computed by `shrink` to input parameters: it shows the expected error and cost when the strategy is executed on an oracle with error parameters diverging from their assumed values. A related issue is the relevance of approximating workers as a random oracle with uniform error over tasks: a threshold effect appears when we try to request arbitrarily high accuracy: when τ is set to a very small values, adding workers did not always provide in practice additional information to complete the most difficult tasks with enough accuracy.

7 Conclusion and Related work

This paper investigates the optimization of queries that filter data using humans. We provided new theoretical insights into the problem, and so designed two novel algorithms – `AdaptSprt` and `shrinkp` – that overcome the scalability issues of previous proposals. We also optimized

algorithms `ladder` and `shrink` from [22], and evaluated thoroughly all algorithms. Our results show that `AdaptSprt` is the only algorithm which performs well for all budgets, while `ladder` performs marginally better for small budgets, but is still extremely slow with larger ones (even when optimized), whereas for moderate budgets our optimized `shrink` works well. With regard to probabilistic strategies, the results show `shrinkp` to have superior reliability, compared to the previous proposals that rely on linear solvers. In summary, our results show that `AdaptSprt` and `shrinkp` both scale well for large budgets. Although cost wise `shrinkp` is optimal, the actual difference of cost is negligible while the running time of `AdaptSprt` is superior.

We already discussed extensively the CrowdScreen framework [22] revisited in this paper. Parameswaran et al. have reviewed in [22] the connections with the related fields in machine learning and statistics, and we thus do not repeat this here and only briefly survey two directions of related work: sequential testing, and classifying with the Crowd.

Sequential tests have been used in numerous fields since their introduction by Wald [23]: quality control, clinical research, acoustic detection, econometrics, etc. Numerous variants have been considered for computing efficient tests, depending on the number of categories tested to which an object may belong; the cost function to be optimized; the form of the strategy boundary [2] and budget constraint [9]; or on whether questions are issued one at a time or in batches [15]. To the best of our knowledge, however, the problem of efficiently computing the optimal test, in the sense studied here, has not yet been addressed. Closest to our work is the system of [10] considering the profit/penalty of correct/wrong answers in a multi-question scenario. Extending our work into such settings is left for future work.

The optimal strategy depends on the query selectivity and the estimated users error. Experiments in [17] stress that classifier performance improves a lot with a proper choice of prior error rates. In practice, the nature of error can be estimated by asking questions to the crowd on a small test set for which the correct answer is already known. Online methods to calculate error rates are discussed in [16], [5], [25] where the error rates are tuned based on comparison of the strategy's decision and the users' answers. One goal of our framework is to avoid any kind of computation online by fixing the filtering strategies beforehand. Adapting strategies according to online error computation is left for further research.

Classification problems with heterogeneous workers and data have been considered in particular in the machine learning literature, exploiting a wide range of techniques from multi-armed bandit problems [1] to singular value decomposition [13], Bayesian learning [24] and variational inference in graphical models [17]. Users and tasks with diverging characteristics raise the challenge of selecting tasks and users to make the most of the budget. For example, Karger et al. [13] propose an algorithm to assign questions to heterogeneous workers with optimal tradeoff between redundancy and accuracy. Empirical models have also been proposed to improve the accuracy of classification by identifying annotation patterns (inherent difficulty of images, groups of users with similar behaviors) [24, 17]. Incorporating some of these ideas in our work is a challenging future work.

Our results focus on binary filters that classify items in two disjoint sets, but can easily be adapted to classify items among n classes, though complexity increases exponentially with n. Devising optimizations to improve performance in this setting is thus a future challenge. Furthermore, processing several filters simultaneously may allow to exploit correlations between filters, or to select dynamically the questions that would be most informative [4, 6, 12].

Finally, empirical studies show that batching tasks may have positive impact on Crowd-Sourcing efficiency [19]. Similarly, pre-recruiting schemes [3], that allow to obtain answers

from the workers within seconds, may help to exploit the full benefit of sequential testing without increasing latency. Devising optimization strategies with batches is challenging.

Acknowledgements. The authors are very thankful to A. Parameswaran for helpful discussions. This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, and by the Israel Ministry of Science.

References

1 Ittai Abraham, Omar Alonso, Vasilis Kandylas, and Aleksandrs Slivkins. Adaptive crowdsourcing algorithms for the bandit survey problem. *To appear in JMLR W&CP*, 30, 2013.

2 T. W. Anderson. A modification of the sequential probability ratio test to reduce the sample size. *The Annals of Math. Stat.*, 31(1):pp. 165–197, 1960. URL: `http://www.jstor.org/stable/2237502`.

3 Michael S. Bernstein, David R. Karger, Robert C. Miller, and Joel Brandt. Analytic methods for optimizing realtime crowdsourcing. In *Collective Intelligence*, 2012.

4 Rubi Boim, Ohad Greenshpan, Tova Milo, Slava Novgorodov, Neoklis Polyzotis, and Wang Chiew Tan. Asking the right questions in crowd data sourcing. In *ICDE*, pages 1261–1264, 2012. `doi:10.1109/ICDE.2012.122`.

5 Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI*, 2010.

6 Nilesh N. Dalvi, Aditya G. Parameswaran, and Vibhor Rastogi. Minimizing uncertainty in pipelines. In *NIPS*, pages 2951–2959, 2012.

7 Jacques Dutka. The incomplete beta function – a historical profile. *Archive for History of Exact Sciences*, 24(1):11–29, 1981. `doi:10.1007/BF00327713`.

8 Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011. `doi:10.1145/1989323.1989331`.

9 Peter Frazier and Angela J. Yu. Sequential hypothesis testing under stochastic deadlines. In *NIPS*, 2007.

10 Jinyang Gao, Xuan Liu, Beng Chin Ooi, Haixun Wang, and Gang Chen. An online cost sensitive decision-making method in crowdsourcing systems. In *SIGMOD*, pages 217–228, 2013. `doi:10.1145/2463676.2465307`.

11 Benoit Groz, Ezra Levin, Isaco Meilijson, and Tova Milo. Filtering with the crowd: Crowdscreen revisited. https://hal.archives-ouvertes.fr/view/index/docid/1239458.

12 Haim Kaplan, Ilia Lotosh, Tova Milo, and Slava Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9):697–708, 2013.

13 David R. Karger, Sewoong Oh, and Devavrat Shah. Efficient crowdsourcing for multi-class labeling. In *SIGMETRICS*, pages 81–92, 2013. `doi:10.1145/2465529.2465761`.

14 Donald E. Knuth. *The Art of Computer Programming, Volume IV, draft of 7.2.1.6*. Addison-Wesley, 2004.

15 Walter Lehmacher and Gernot Wassmer. Adaptive sample size calculations in group sequential trials. *Biometrics*, 55(4):1286–1290, 1999. `doi:10.1111/j.0006-341X.1999.01286.x`.

16 Christopher H. Lin, Mausam, and Daniel S. Weld. Dynamically switching between synergistic workflows for crowdsourcing. In *AAAI*, 2012.

17 Qiang Liu, Jian Peng, and Alexander T. Ihler. Variational inference for crowdsourcing. In *NIPS*, pages 701–709, 2012.

18 Adam Marcus, David R. Karger, Samuel Madden, Rob Miller, and Sewoong Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

19 Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

20 Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

21 Aditya G. Parameswaran. Personal Communication.

22 Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012. doi:10.1145/2213836.2213878.

23 A. Wald. Sequential tests of statistical hypotheses. *The Annals of Math. Stat.*, 16(2):pp. 117–186, 1945. URL: http://www.jstor.org/stable/2235829.

24 Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The multidimensional wisdom of crowds. In *NIPS*, pages 2424–2432, 2010.

25 Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, pages 2035–2043, 2009.

Streaming Partitioning of Sequences and Trees*

Christian Konrad

School of Computer Science, Reykjavík University, Reykjavík, Iceland
christiank@ru.is

Abstract

We study streaming algorithms for partitioning integer sequences and trees. In the case of trees, we suppose that the input tree is provided by a stream consisting of a depth-first-traversal of the input tree. This captures the problem of partitioning XML streams, among other problems.

We show that both problems admit deterministic $(1+\epsilon)$-approximation streaming algorithms, where a single pass is sufficient for integer sequences and two passes are required for trees. The space complexity for partitioning integer sequences is $O(\frac{1}{\epsilon} p \log(nm))$ and for partitioning trees is $O(\frac{1}{\epsilon} p^2 \log(nm))$, where n is the length of the input stream, m is the maximal weight of an element in the stream, and p is the number of partitions to be created.

Furthermore, for the problem of partitioning integer sequences, we show that computing an optimal solution in one pass requires $\Omega(n)$ space, and computing a $(1+\epsilon)$-approximation in one pass requires $\Omega(\frac{1}{\epsilon} \log n)$ space, rendering our algorithm tight for instances with $p, m \in O(1)$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Streaming Algorithms, XML Documents, Data Partitioning, Communication Complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.13

1 Introduction

Partitioning Massive Data Sets. *Data partitioning* is a widely employed technique for processing massive data sets. The input data is partitioned into (not necessarily disjoint) subsets of much smaller sizes which are then distributed to different computational units. Parallel or distributed algorithms are then executed on the partitioned data.

The data partitioning step can be a difficult task in itself, especially if the data sets considered are massive. Some partitioning problems are NP-hard (some are even hard to approximate [3]), while others are more amenable. However, in the context of massive data sets, it is not clear whether even the more amenable problems can be solved efficiently.

In this paper, we are therefore interested in how well big data sets can be partitioned by massive data set algorithms. We focus on streaming algorithms, and we consider the problems of partitioning integer sequences and partitioning trees. Streaming algorithms use a small random access memory which is usually only of poly-logarithmic size in the input. They scan the entire data from left to right sequentially in passes and therefore make optimal use of data locality.

Partitioning Integer Sequences. Let $X \in \{0, \ldots, m\}^n$ be a sequence of integers of length n, for an integer m. Given an integer parameter p, the goal is to partition X into p contiguous blocks so as to minimize the maximum weight (sum of elements) of a block. In other words, we have to find $p-1$ separators $s_1, \ldots, s_{p-1}$ with $1 = s_0 \leq s_1 \leq \cdots \leq s_{p-1} \leq s_p = n+1$ such

* Supported by Icelandic Research Fund grants 120032011 and 152679-051.

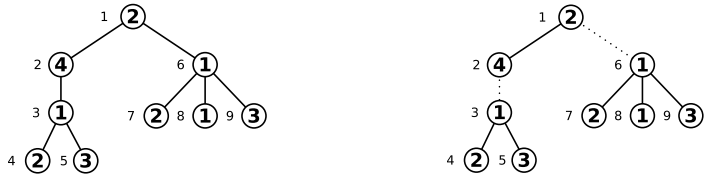


Figure 1 *Left*: A weighted tree t. The small numbers next to the nodes denote their IDs. The trace of a depth-first-traversal of t is $241\overline{2}\overline{3}3\overline{1}\overline{4}12\overline{2}\overline{1}\overline{1}3\overline{3}\overline{1}\overline{2}$, where we ambiguously write w for ('d', w) and $\overline{w}$ for ('u', w). *Right*: Partitioning of t into three parts with bottleneck value 7. A streaming algorithm should output the IDs of the root nodes of the created partitions. Here, these are $1, 3, 6$.

that $\max\left\{\sum_{i=s_j}^{s_{j+1}-1} X_i \;\middle|\; j \in \{0,\ldots,p-1\}\right\}$ is minimized. We will abbreviate this problem as Part.

This partitioning problem appears in many applications, especially in the context of load balancing, and has been extensively studied both from a theoretical [6, 9, 11, 20, 21, 13, 10] and a practical perspective [22, 25]. One example application is the decomposition of large computational meshes along space-filling curves [26, 15, 4] in parallel scientific computing, where multi-dimensional grid elements are ordered with respect to a traversal along a space-filling curve, giving rise to the one-dimensional problem of partitioning integer sequences. Very efficient exact algorithms for this problem exist, for example the $O(n \log n)$ time algorithm of Khanna et al. [13], the $O(n + p^{1+\epsilon})$ time algorithm of Han et al. [10], and a highly non-trivial optimal $O(n)$ time algorithm of Frederickson [9].

Partitioning Trees. The problem of partitioning integer sequences is extended to trees as follows: Given a rooted unranked node-weighted tree t with weights taken from the set $\mathcal{U} = \{0,\ldots,m\}$ for an integer m, and an integer p, the goal is to partition t into p subtrees $t_1,\ldots,t_p$ by removing $p-1$ edges so as to minimize the maximum weight of a subtree. If we are allowed random access to the input, this problem can also be solved optimally by an $O(n)$ time algorithm by Frederickson [9]. Classical applications of this problem include paging and overlaying [24]. More importantly, this problem also captures the problem of partitioning XML documents (see further below), which is the main motivation of this work.

In the case of integer sequences, we assume that the input stream for our streaming algorithm is the integer sequence X itself. In the case of trees, since we target XML documents, we assume that the input stream constitutes the trace of a depth-first-traversal of the input tree t. More precisely, the input stream is the following trace $X \in (\{\text{'d'},\text{'u'}\} \times \mathcal{U})^{2n}$ of a depth-first-traversal of t ('d' stands for down-step and 'u' stands for up-step): If a node with weight $w \in \mathcal{U}$ is visited top-down (bottom-up) at step i then $X_i = (\text{'d'}, w)$ (resp. $X_i = (\text{'u'}, w)$). We also say that a node v of the input tree t has ID i if it is the ith node that is visited by the depth-first-traversal. The trace X is fed into our streaming algorithm, and we require that the algorithm outputs the IDs of the root nodes of the p partitions (or subtrees). We will denote this problem by Tree. In Figure 1, we give an example tree illustrating the trace of a depth-first-traversal as well as a partitioning of the example tree.

Partitioning XML Documents and Other Applications. The traces of depth-first-traversals of trees constitute a Dyck language, and, vice versa, every word of a Dyck language can be seen as the concatenation of traces obtained from depth-first-traversals of the trees of a forest. Dyck languages are languages of well-parenthesized expressions, and algorithms that process Dyck languages such as [29, 19] and our work therefore have potential applications

in all areas where those types of expressions appear. These include arithmetic expressions, the sequence of `CALL` and `RETURN` instructions of the traces of program executions, and XML documents, which are probably the most relevant application for the database community. An XML document can be seen as the trace of a depth-first-traversal of its underlying document tree.

Partitioning XML documents is for instance widely used in the area of parallel XML databases and for the parallel evaluation of queries such as XPath, where the term *XML fragmentation* is usually employed [7]. In this context, additional constraints on the partitioning other than load balancing are often imposed in order to avoid data dependencies between different partitions, however, achieving good load balancing is crucial for any performant solution. Kim and Kang [14] study the setting where an XML document is fragmented and the resulting partitions are streamed to mobile client devices, which then distributively evaluate a query. They underline the importance of small partitions by pointing out that: "No matter how efficient a fragmentation method is, it is useless and impractical unless the size of every resulting fragment is less than that of the buffer allocated for receiving the fragments at the client devices." Thus, our work can be seen as a first step towards streaming XML fragmentation. Further potential applications of the tree partitioning problem include the parallel validation of XML documents with respect to local validity schemata such as DTDs [16] and similar problems that require only local knowledge of an XML document.

Starting Point: Parametric Search. Many previous works tackle Part and Tree using *parametric search* [9, 10, 13]. For both problems, given a value B, there is an algorithm that computes a partitioning in linear time with bottleneck value at most B if such a partitioning exists. If there is no such partitioning, then the algorithm fails. Such an algorithm can be regarded as a feasibility tester: Given a value B, it checks whether the bottleneck value of an optimal partitioning B^* is larger than B (B is not feasible) or smaller than/equal to B (B is feasible). A trivial range for B^* is $\{0, 1, \ldots, mn\}$[1] since the input consists of n integers or tree nodes of maximal weight m. Thus, via a binary search, running the feasibility tester $\mathrm{O}(\log(mn))$ times, an exact algorithm with runtime $\mathrm{O}(n \log(mn))$ can be obtained.

For an integer sequence X, a value B can be tested by traversing X from left to right, setting up partitions of maximal sizes at most B. If at most p partitions are created, then B passes the test, otherwise it fails. This tester, denoted Probe in the literature, can be implemented as a one-pass streaming algorithm with $\mathrm{O}(p \log(n) + \log(mn))$ space[2]. Hence, using the previous binary search, a $\mathrm{O}(\log(mn))$ passes, $\mathrm{O}(p \log(n) + \log(mn))$ space exact streaming algorithm for Part can be obtained. For Tree, a linear time feasibility tester also exists [9], however, it appears impossible to implement it space efficiently as a one-pass streaming algorithm.

Probe can also be used to obtain a one-pass streaming algorithm with approximation factor $(1+\epsilon)$ (meaning that a partitioning with bottleneck value at most by a factor $(1+\epsilon)$ larger than the optimal bottleneck value is computed): In one pass, we run multiple copies of Probe testing values $(1+\epsilon)^i$, for all integers i with $0 \leq i \leq \frac{\log(mn)}{\log(1+\epsilon)} = \mathrm{O}(\frac{\log(mn)}{\epsilon})$, in parallel, and we return the partitioning with the smallest feasible bottleneck value. We observe that only $\mathrm{O}(\frac{\log p}{\epsilon})$ copies of Probe are active at any moment during the processing of the stream:

[1] A slightly better upper bound for B^* is $\lceil \frac{mn}{p} \rceil + m$, however, this does not change our arguments and only complicates the presentation.

[2] $\mathrm{O}(p \log(n))$ for storing $p-1$ partition boundaries, and $\mathrm{O}(\log(mn))$ for accumulating the weight of the current partition.

Let B' be the smallest bottleneck value that has not yet been declared infeasible by a copy of Probe. Then, the total weight of the prefix of the stream seen so far is at most pB', and hence the copies of Probe testing values larger than pB' have not yet set their first partition boundaries. Thus, it is not necessary to explicitly execute them yet. Using this observation, the following result can be obtained:

▶ **Fact 1.** *There is a one-pass* $(1+\epsilon)$*-approximation streaming algorithm for* Part *with space* $O(\frac{\log p}{\epsilon} \cdot (p \log(n) + \log(mn)))$.

We regard this algorithm as a baseline against which we will compare our results. Its main weakness is its update time [3]: Since $\Theta(\frac{\log p}{\epsilon})$ copies of Probe have to be updated, the worst-case and amortized update time is $\Theta(\frac{\log p}{\epsilon})$. Furthermore, the $\log p$ factor in the space complexity of the algorithm seems unnatural. In this paper, we will show that, using a very different technique, both weaknesses can be overcome.

Results and Techniques. In many areas of computer science, a common technique for dealing with large objects is to replace them with smaller ones that capture important properties of the initial object sufficiently well. Examples include kernelization [18], approximate distance oracles [27, 23], and graph sparsification [5]. In recent years, this technique has proved useful for data streaming algorithms, e.g., [8, 1, 2, 12], and we follow this line here.

Our algorithms for Part and Tree compute much smaller instances from the problem instance described in the input stream, so that a $(1+\epsilon)$-approximate partitioning can be deduced from an exact partitioning of the small instances. Our contribution is two-fold: First, we identify the right properties that guarantee that the smaller objects still capture $(1+\epsilon)$-approximate partitionings of the original instance. In the case of Part, we prove that a small instance of size $O(\frac{p}{\epsilon})$ is sufficient, and for Tree, an instance of size $O(\frac{p^2}{\epsilon})$ suffices. Note that, in both cases, the size is independent of n. Second, we prove that the small objects with the right properties can be computed space efficiently in the streaming model. Then, in a post-processing step, we use exact algorithms for partitioning the small instances.

This technique leads to the following algorithmic results:

- A deterministic one-pass $(1+\epsilon)$-approximation streaming algorithm for Part with space $O(p \log(mn)/\epsilon)$, worst case update time $O(1)$, and post-processing time $O(p/\epsilon)$ (**Theorem 7**).
- A deterministic two-pass $(1+\epsilon)$-approximation streaming algorithm for Tree with space $O(p^2 \log(mn)/\epsilon)$, worst case update time $O(1)$ and post-processing time $O(p^2/\epsilon)$ (**Theorem 15**).

Note that our algorithm for Part improves on the space complexity and the update time of the algorithm described in Fact 1. Last, we complement our algorithms with lower bounds for Part obtained through results in communication complexity:

- Via a reduction to the one-way two-party communication problem Index, we show that computing an exact solution for Part in one pass requires $\Omega(n)$ space (**Theorem 18**).
- Via a more involved combinatorial argument, we show that algorithms that compute a $(1+\epsilon)$-approximation for Part require space $\Omega(\log(n)/\epsilon)$ (**Theorem 20**).

The latter lower bound shows that our algorithm for Part is best possible for $p, m \in O(1)$, and, in particular, that the $\frac{1}{\epsilon}$ factor in the space complexity is unavoidable for one-pass algorithms.

[3] The time between two consecutive read operations on the stream

Outline. We consider PART in Sec. 3 and TREE in Sec. 4. Our lower bounds are given in Sec. 5, and we conclude in Sec. 6. Due to space restrictions, the proofs of lemmas and theorems marked by **(*)** are omitted.

2 Notations and Definitions

Streaming Algorithms. Generally, we denote the input stream for our streaming algorithms by $X = X[1], X[2], \ldots$. In the case of integer sequences, the length of X is n, and in the case of trees, the length is $2n$, since each of the n nodes of the input tree is visited twice by a depth-first-traversal. A *streaming algorithm* that processes X reads the elements of X sequentially one-by-one in passes. Streaming algorithms are defined as follows:

▶ **Definition 1** (Streaming algorithm)**.** An algorithm **A** is a $p(n)$-pass deterministic/randomized *streaming algorithm* with memory $s(n)$, update time $t(n)$, amortized update time $a(n)$, and post-processing time $t'(n)$, if for every input stream X of length n:

1. **A** performs at most $p(n)$ passes over the input stream X,
2. **A** maintains a random access memory of size $s(n)$,
3. **A** has worst case running time $t(n)$ between two consecutive read operations,
4. **A** has average running time $a(n)$ between two consecutive read operations,
5. **A** has running time $t'(n)$ between the last read operation and the output of the result.

If **A** is randomized then **A** has access to an infinite number of independent random coin flips, and it outputs a correct solution with probability at least $2/3$.

The algorithms presented in this paper are deterministic, however, we include randomized algorithms in Definition 1 since our lower bounds also hold for randomized algorithms. Furthermore, we suppose that reading an element from the input stream takes $O(1)$ time.

Notations for Sequences. For an array X, we denote the ith element by either $X[i]$ or X_i. For $i \leq j$, the array consisting of $X[i], X[i+1], \ldots, X[j]$ is denoted by $X[i, j]$.

Notations for Trees. Let t be an unranked rooted tree on n nodes. We denote by $\mathrm{rt}(t)$ the root of t. For any node $v \in t$, we denote by $\mathrm{stree}_t(v)$ the subtree of t rooted at v. For two nodes $x, y \in t$, let $\mathrm{lca}_t(x, y)$ denote the lowest common ancestor of x and y in t, and for a subset of nodes $U \subseteq t$, let $\mathrm{lca}_t(U)$ denote the lowest common ancestor of all nodes of U in t (if $U = \{u\}$ then $\mathrm{lca}_t(U) = u$). Given a node $v \in t$, we denote its ID, i.e., the position of v with respect to a depth-first-traversal of t, by $\mathrm{Id}_t(v)$. Given an ID i, we denote its corresponding node in t by $\mathrm{node}_t(i)$. With this notation, we have $\mathrm{node}_t(\mathrm{Id}_t(v)) = v$.

3 Algorithm For Partitioning Sequences

The main idea of our algorithm is the computation of a *coarse version* Y of the input stream X. Intuitively, a coarse version is obtained from X by repeatedly merging subsequent integers into a single element whose weight is the sum of the merged elements. If the coarse version Y fulfills certain properties, it can be shown that, from an optimal partitioning of Y, a $(1+\epsilon)$-approximation to the optimal partitioning of X can be obtained.

In Subsec. 3.1, we define coarse versions. Then, in Subsec. 3.2, we show how an appropriate coarse version Y can be computed in one pass. In a post-processing step, the coarse version Y is partitioned optimally, giving a $(1+\epsilon)$-approximation to the optimal partitioning of X.

3.1 Coarse Version

We now define a c-coarse version of X, and we prove that an optimal partitioning of Y provides an approximate partitioning of X.

▶ **Definition 2** (c-coarse version)**.** Let $m, m', n, n' \in \mathbb{N}$. Let $X \in \{0, \ldots, m\}^n$ and $Y \in \{0, \ldots, m'\}^{n'}$ be integer sequences. Then Y is a *c-coarse version* of X if $n' \leq n$ and there is a mapping $f : \{1, \ldots, n\} \to \{1, \ldots, n'\}$, denoted the *coarsening function*, such that:

1. f is surjective and increasing,
2. For every $1 \leq i' \leq n' : \sum_{i \in f^{-1}(i')} X[i] = Y[i']$,
3. For every $1 \leq i' \leq n' : Y[i'] \leq X[\min f^{-1}(i')] + c$.

Suppose that $X = 1\,3\,2\,1\,1\,5$. Then, $4\,4\,5$ is a 3-coarse version of X where the grouping of the elements of X has been done as follows: $\mathbf{1}\,3 \,|\, \mathbf{2}\,1\,1 \,|\, \mathbf{5}$. In order to fulfill Item 3 of the previous definition, the bold elements of the previous grouping have to be mapped to elements in the 3-coarse version that are larger by at most 3.

Lemma 3 shows that an optimal partitioning of a c-coarse version has a bottleneck value that is at most by the additive term c larger than the bottleneck value of an optimal partitioning of the initial sequence.

▶ **Lemma 3.** *Let $m, m', n, n' \in \mathbb{N}$. Let $X \in \{0, \ldots, m\}^n$ be an integer sequence and let $Y \in \{0, \ldots, m'\}^{n'}$ be a c-coarse version of X with coarsening function f, for a parameter c. Let B^* be the bottleneck value of an optimal partitioning of X into p parts. Let $s'_0, s'_1, \ldots, s'_p$ be the separators of an optimal partitioning of Y into p parts and let B'^* be the bottleneck value. Then:*

1. *The separators $s_0, s_1, \ldots, s_{p-1}, n+1$ with $s_i = \min f^{-1}(s'_i)$ for $i = 0, \ldots, p-1$ induce a partitioning of X with bottleneck value B'^*,*
2. *$B'^* \leq B^* + c$.*

Proof. Concerning Item 1, it follows from Item 2 of Definition 2 that the weight of the partition induced by separators s'_i and s'_{i+1} in Y equals the weight of the partition induced by separators s_i and s_{i+1} in X, for every i. Therefore, the bottleneck value is also the same.

Concerning Item 2, consider an optimal partitioning of X into p parts with bottleneck value B^*, and let $s^*_0, \ldots, s^*_p$ denote the separators of this partitioning. We will argue that, given the s^*_i, we can compute a partitioning of Y with separators $\tilde{s}_i$ with bottleneck value at most $B^* + c$. Since the optimal partitioning of Y with separators $s'_0, s'_1, \ldots, s'_p$ is at least as good, the result follows.

We define $\tilde{s}_p = n' + 1$, and for $i = 0, \ldots p-1$, let $\tilde{s}_i = \min\{j : \min f^{-1}(j) \geq s^*_i\}$, i.e., partition i in Y starts with the first element whose pre-image starts in partition i in X.

Now we argue that for any i, the weight of partition i in Y is at most the weight of partition i in X plus c. This then proves Item 2, as this property also holds for the bottleneck partition. We have:

$$\sum_{j=\tilde{s}_i}^{\tilde{s}_{i+1}-1} Y[j] = \left(\sum_{j=\tilde{s}_i}^{\tilde{s}_{i+1}-2} Y[j]\right) + Y[\tilde{s}_{i+1}-1] \leq \left(\sum_{j=\min f^{-1}(\tilde{s}_i)}^{\max f^{-1}(\tilde{s}_{i+1}-2)} X[j]\right) +$$

$$+ \big(X[\min f^{-1}(\tilde{s}_{i+1}-1)] + c\big) = \left(\sum_{j=\min f^{-1}(\tilde{s}_i)}^{\min f^{-1}(\tilde{s}_{i+1}-1)} X[j]\right) + c \leq \left(\sum_{j=s^*_i}^{s^*_{i+1}-1} X[j]\right) + c.$$

For the first inequality, we used Item 2 of Definition 2 to rewrite the sum, and Item 3 of Definition 2 in order to bound $Y[\tilde{s}_{i+1}-1]$. For the second inequality, we extended the

Algorithm 1 Computation of coarse version

Require: Number of partitions p, parameter ϵ for $(1+\epsilon)$-approximation

$s \leftarrow 4\lceil \frac{p}{\epsilon} \rceil$ {Size of array Y}
$Y \leftarrow (Y_{i1}, Y_{i2})_i$ array of integer tuples of length s, initially all tuples are $(0,0)$
while input stream not empty **do**
  $k \leftarrow (\arg\min_{1 \le i \le s} \{Y_{i1} = 0\}) - 1$
  $x_1 \dots x_{s-k} \leftarrow$ next $s-k$ integers from input stream, if fewer than $s-k$ integers are left then interpret the missing ones as 0s
  $\forall i \in \{1, \dots, s-k\} : Y[k+i] \leftarrow (x_i, 0)$ {Append the x_i to Y}
  $S \leftarrow \sum_{i=1}^{s} Y_{i1} + Y_{i2}$ {Weight of Y, equals weight of input stream seen so far}
  $Y \leftarrow \text{CPROBE}(Y, \lfloor S\epsilon/p \rfloor)$
end while
$Y' \leftarrow$ integer array of length s with $\forall 1 \le i \le s : Y'[i] = Y_{i1} + Y_{i2}$
return Y'

Algorithm 2 CPROBE(Y, c)

Require: $Y = (Y_{i1}, Y_{i2})_i$ array of integer tuples of length s, integer c

$Z \leftarrow (Z_{i1}, Z_{i2})_i$ array of integer tuples of length s
$j \leftarrow 1$ {current element in Z}
$Z_j \leftarrow Y_1$
for $i = 2 \dots s$ **do**
  if $Z_{j2} + Y_{i1} + Y_{i2} \le c$ **then**
    $Z_{j2} \leftarrow Z_{j2} + Y_{i1} + Y_{i2}$
  else
    $j \leftarrow j+1$, $Z_j \leftarrow Y_i$
  end if
end for
return Z

sum using the observations $s_i^* \le \min f^{-1}(\tilde{s}_i)$, and $s_{i+1}^* - 1 \ge \min f^{-1}(\tilde{s}_{i+1} - 1)$. These observations follow from the definition of $\tilde{s}_i$. ◀

We conclude that in order to obtain a $(1+\epsilon)$-approximation, a $(S\epsilon/p)$-coarse version of X with $S = \sum_{i=1}^{n} X[i]$ is required.

▶ **Corollary 4.** *Let Y be a $(S\epsilon/p)$-coarse version of X where $S = \sum_{i=1}^{n} X[i]$, for some $\epsilon > 0$. An optimal partitioning of Y allows us to obtain a $(1+\epsilon)$-approximation to* PART *on X.*

Proof. Let B^* be the bottleneck value of an optimal partitioning of X. Clearly, $B^* \ge S/p$. By Lemma 3, we obtain the approximation factor: $\frac{B^* + S\epsilon/p}{B^*} = 1 + \frac{S\epsilon/p}{B^*} \le 1 + \frac{S\epsilon/p}{S/p} = 1+\epsilon$. ◀

3.2 Computing Coarse Versions

Algorithm. Our algorithm for computing a $(S\epsilon/p)$-coarse version of the input stream X with $S = \sum_{i=1}^{n} X[i]$ is depicted in Algorithm 1. It uses the subroutine CPROBE (CoarsePROBE) which is depicted in Algorithm 2. CPROBE is similar in spirit to the PROBE algorithm mentioned in the introduction and hence carries a similar name.

Representation of the Coarse Version. Algorithm 1 operates on an array $Y = (Y_{i1}, Y_{i2})_i$ of length $s = 4\lceil \frac{p}{\epsilon} \rceil$ consisting of tuples of integers. Throughout the algorithm, Y will represent

a $(S\epsilon/p)$-coarse version of the already seen prefix of the input integer sequence X, where S is the total weight of the already seen prefix. The coarse version can be explicitly computed from Y as in Line 10 of the algorithm: Replace every tuple (Y_{i1}, Y_{i2}) by the sum $Y_{i1} + Y_{i2}$.

The first element Y_{i1} of every non-zero tuple (Y_{i1}, Y_{i2}) will always equal a value $X[j]$, for some j, and Y_{i2} will always equal $\sum_{l=j+1}^{j'} X[l]$, for some $j' \geq j$. Such a tuple corresponds to the merging of the elements $X[j], \ldots, X[j']$ into a single element in the coarse version. We store the first element $X[j]$ and the remaining elements $\sum_{l=j+1}^{j'} X[l]$ separately as Y_{i1} and Y_{i2} in order to be able to guarantee that the crucial Item 3 of Definition 2 of a c-coarse version is fulfilled, namely, $Y_{i2} \leq c$, for some c.

Description of the Algorithm. In the first iteration of the while-loop, Y is filled with the first s integers of the input stream so that $Y = (X_1, 0), (X_2, 0), \ldots, (X_s, 0)$. CPROBE is then invoked on Y and parameter $\lfloor S\epsilon/p \rfloor$, where S is the current total weight of Y, i.e., $S = \sum_{i=1}^{s} Y_{i1} + Y_{i2}$ (or, equivalently, the weight of the prefix of the input stream seen so far). CPROBE(Y, c) computes an array of tuples $Z = (Z_{i1}, Z_{i2})_i$ representing a c-coarse version of Y. It greedily merges adjacent tuples $(Y_{i1}, Y_{i2}), \ldots, (Y_{j1}, Y_{j2})$ into a tuple $(Z_{k1}, Z_{k2}) = (Y_{i1}, Y_{i2} + \sum_{l=i+1}^{j} Y_{l1} + Y_{l2})$ for some k, where j is the largest value such that Z_{k2} does not exceed c. Note that the first parameter Z_{k1} takes the value of Y_{i1} unaltered. This guarantees that, throughout the algorithm, the first parameter of any non-zero tuple (Y_{i1}, Y_{i2}) always equals a value of the input stream $X[j]$, for some j.

We will prove in Lemma 5 that the length of the output sequence Z of CPROBE is at most $3p/\epsilon + 1$. Since Y is of length $4\lceil p/\epsilon \rceil$, in the next iteration of the while-loop of Algorithm 1, at least $p/\epsilon - 1$ new elements from the input stream are added to Y, which guarantees progress in every iteration of the while-loop. The process continues until the entire input stream has been processed.

Analysis. In the following, we will denote the input stream by $X \in \{1, \ldots, m\}^n$. For simplicity, we assume that $X[i] \gneq 0$, for all i, which is not a restriction since 0s in the input stream could simply be skipped by the algorithm.

▶ **Lemma 5.** *Consider the state of variable Y of Algorithm 1 at the beginning of iteration w of the while-loop, for any $w \geq 1$. Suppose that the prefix $X[1, q]$ of the input stream has been processed up until this point. Furthermore, let k be the largest j such that $Y[j] \neq (0, 0)$, and let $S = \sum_i Y_{i1} + Y_{i2}$. Then:*

1. $k \leq \frac{3p}{\epsilon} + 1$,
2. $Y_{i2} \leq S\epsilon/p$ *for every* $1 \leq i \leq s$,
3. *If $w \geq 2$, then there are integers $1 = t_1 < t_2 < \cdots < t_k < t_{k+1} = q + 1$ so that*

$$Y[1, k] = \left(X[t_1], \sum_{l=t_1+1}^{t_2-1} X[l]\right), \left(X[t_2], \sum_{l=t_2+1}^{t_3-1} X[l]\right), \ldots, \left(X[t_k], \sum_{l=t_k+1}^{t_{k+1}-1} X[l]\right).$$

Proof. We prove the statement by induction. Consider the first iteration. Then, all tuples of Y are $(0, 0)$ and $X[1, q]$ is an empty sequence, and the lemma is trivially true.

Now, suppose that the lemma is true in iteration $w \geq 1$. We will prove that it also holds in iteration $w + 1$.

Let $q' = q + (s - k - 1)$. Then, $Y[k + 1, s] = X[q + 1, q']$ after Line 6 of the algorithm (if $q' > n$ then suppose that a sequence of 0s follows the input stream). Let S' be the total weight of Y after the execution of Line 6 of the algorithm. Clearly, Item 3 is still fulfilled after appending elements of X to Y in Line 6. The left-to-right processing of Y

in CProbe guarantees that only consecutive elements are merged and that the order of Y stays intact. Furthermore, the first parameters of the tuples are copied (in Lines 3 and 8) and are therefore never changed, proving Item 3. The if-statement in Line 5 guarantees that merged elements do not exceed the value of $c = \lfloor S'\epsilon/p \rfloor$, thus ensuring Item 2. Let k' be the largest index such that $Y[k'] \neq (0,0)$, after the run of CProbe. To prove Item 1, notice that after the run of CProbe, for any $(Y_{i1}, Y_{i2}), (Y_{i+1,1}, Y_{i+1,2})$, we have $Y_{i2} + Y_{i+1,1} + Y_{i+1,2} > \lfloor S'\epsilon/p \rfloor$, since otherwise $(Y_{i+1,1}, Y_{i+1,2})$ would have been added to Y_{i2} in the algorithm. Thus, $S' = \sum_{i=1}^{k'} Y_{i1} + Y_{i2} > \frac{k'-1}{2} \cdot \lfloor S'\epsilon/p \rfloor$, which implies

$$k' < 1 + \frac{2S'p}{S'\epsilon - p} = 1 + \frac{2p}{\epsilon} + \frac{2p^2}{\epsilon(S'\epsilon - p)} \leq 1 + \frac{2p}{\epsilon} + \frac{2p^2}{\epsilon(\lceil 4\frac{p}{\epsilon} \rceil \epsilon - p)} < 1 + \frac{3p}{\epsilon},$$

where we used $S' \geq 4\lceil \frac{p}{\epsilon} \rceil$, since this quantity equals the length of Y, and all first elements of the tuples of Y are at least 1. ◀

▶ **Lemma 6.** *Algorithm 1 is a deterministic one-pass streaming algorithm that computes a $(S\epsilon/p)$-coarse version of the input stream $X \in \{1, \ldots, m\}^n$ using space $O(p \log(mn)/\epsilon)$, where $S = \sum_i X[i]$. It can be implemented with worst case update time $O(1)$.*

Proof. The structural properties (2) and (3) of Lemma 5 guarantee that the returned integer sequence Y' is a $(S\epsilon/p)$-coarse version of the input stream X and thus establish correctness of the algorithm. Property 1 ensures that the algorithm makes progress in every iteration: Since at most $\frac{3p}{\epsilon} + 1$ tuples of Y are different from $(0,0)$, in every iteration at least $s - (\frac{3p}{\epsilon} + 1) \geq \frac{p}{\epsilon} - 1$ integers from the input sequence are consumed.

Note that the algorithm as it is implemented in Algorithm 1 has amortized update time $O(1)$. Indeed, in every iteration of the while-loop, $\Theta(\frac{p}{\epsilon})$ integers from the input stream are consumed, and the runtime of one iteration is $O(\frac{p}{\epsilon})$. Using the following standard trick, we go from amortized update time to worst case update time: While executing the algorithm, we simultaneously read integers from the input stream into a buffer of size $\Theta(s)$ one at a time every $O(1)$ operations. Then, instead of consuming integers from the input stream in Line 5 of the algorithm, we consume the integers from the buffer. ◀

Using Lemma 6, we compute a $(S\epsilon/p)$-coarse version of the input stream, and we split it optimally in a post-processing step. This yields our main result of this section.

▶ **Theorem 7.** *There is a deterministic one-pass $(1+\epsilon)$-approximation streaming algorithm for Part with space $O(p \log(mn)/\epsilon)$, worst case update time $O(1)$, and $O(p/\epsilon)$ post-processing time.*

Proof. First, we will use Algorithm 1 in order to compute a coarse version of the input stream X. Let Y be the generated output which, according to Lemma 6, is a $(S\epsilon/p)$-coarse version of X. Then, we partition Y optimally using the exact linear time algorithm of Frederickson [9]. Following Lemma 3, we compute a $(1+\epsilon)$-approximation to an optimal partitioning of X.

Note that in order to deduce a partitioning of X from a partitioning of Y, it is required that every $Y[j]$ know the value $\min f^{-1}(j)$, where f is the coarsening function. This can be ensured by annotating the elements of the stream $X[i]$ by its position in the stream i, and forwarding those annotations to Y. ◀

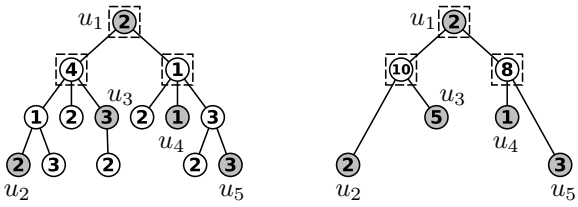


Figure 2 *Left*: An example tree. The highlighted nodes constitute set U of Definition 8. Set L of Definition 8 consists of the nodes within boxes. Note that the root node u_1 is by definition in B, however, it is also in L since $\mathrm{lca}_t(u_3, u_4) = u_1$. *Right*: The structure tree $\mathrm{ST}(U)$.

4 Algorithm for Partitioning Trees

We now present our algorithm for Tree. Our algorithm computes a *coarse structure tree*, i.e., a tree with much fewer nodes than the input tree t that captures the structure of t well enough so that an optimal partitioning of the structure tree allows us to obtain a $(1+\epsilon)$-approximation for Tree on t. In Subsec. 4.1, we define structure trees and we prove that an optimal partitioning of an appropriate structure tree approximates an optimal partitioning of t within a factor of $1+\epsilon$. This result is the most technical contribution of this paper. Then, in Subsec. 4.2, we argue that an appropriate structure tree can be computed with $\mathrm{O}(\frac{p^2}{\epsilon}\log(mn))$ space in the streaming model.

4.1 Structure Trees

Let t denote the weighted input tree that is described by the input stream X. Let the weight function of t be ω so that $\omega(v)$ denotes the weight of node $v \in t$, and let $\omega(t') := \sum_{v \in t'} \omega(v)$, for a subtree $t' \subseteq t$. Our definition of structure trees is as follows:

▶ **Definition 8** (Structure Tree). Let $U = \{u_1, \ldots, u_k\} \subseteq t$ be a set of *breakpoints* (nodes) ordered with respect to a depth-first-traversal of t, and suppose that $\mathrm{rt}(t) \in U$ (which implies that $u_1 = \mathrm{rt}(t)$). Let $L = \{\mathrm{lca}_t(u_i, u_{i+1}) \mid 1 \leq i \leq k-1\}$. Then the *structure tree* $\mathrm{ST}(U)$ of t with respect to U is a weighted tree with weight function ω' on vertex set $U \cup L$ such that there is an edge between $x, y \in U \cup L$ iff among the nodes $U \cup L$, y is the lowest ancestor of x in t. For a node $x \in U \cup L$, let $D(x) \subseteq t$ denote the set of nodes such that x is the lowest ancestor among the nodes $U \cup L$. Then, we define $\omega'(x) = \sum_{y \in D(x)} \omega(y)$.

Definition 8 is illustrated in Figure 2. An immediate consequence of the definition of a structure tree is that for every node $v \in \mathrm{ST}(U)$, the weight of the subtree rooted at v in $\mathrm{ST}(U)$ is the same as the weight of the subtree rooted at v in t. Consider a partitioning S of $\mathrm{ST}(U)$ (i.e., the roots of the subtrees induced by the partitioning). Since the weight of $\mathrm{stree}_{\mathrm{ST}(U)}(v)$ of any node $v \in \mathrm{ST}(U)$ is the same in tree t, clearly the bottleneck value of a partitioning S of $\mathrm{ST}(U)$ equals the bottleneck value of the partitioning S applied on t. This observation is similar to Item 1 of Lemma 3 for sequences, and is summarized in the following fact.

▶ **Fact 2.** *Let S be a partitioning of $\mathrm{ST}(U)$ with bottleneck value B. Then, S is also a partitioning of t with bottleneck value B.*

Choosing Good Breakpoints. The set of breakpoints U on which the structure tree $\mathrm{ST}(U)$ is built determines how well $\mathrm{ST}(U)$ represents the input tree t and thus how well a partitioning

of $\mathrm{ST}(U)$ approximates a partitioning of t. We will choose set U according to the following definition, which establishes a bridge between trees and integer sequences:

▶ **Definition 9** (c-coarse Structure Tree)**.** Let $V = \{v_1, \ldots, v_n\}$ denote the nodes of the input tree t ordered with respect to a depth-first-traversal of t. Let $X'[i] = \omega(v_i)$. For $U \subseteq V$ we say that $\mathrm{ST}(U)$ is a *c-coarse structure tree* of t if there exists a c-coarse version Y of X' of length $n' \leq n$ and coarsening function f such that $v_i \in U$ iff $\exists 1 \leq i' \leq n' : \min f^{-1}(i') = i$.

Consider the example tree in Figure 2. Then, we have $X' = 2\,4\,1\,2\,3\,2\,3\,2\,1\,2\,1\,3\,2\,3$. A 5-coarse version of X' is $7\,7\,6\,7\,3$ where X' has been grouped as follows: $\mathbf{2}\,4\,1\,|\,\mathbf{2}\,3\,2\,|\,\mathbf{3}\,2\,1\,2\,|\,\mathbf{1}\,3\,2\,|\,\mathbf{3}$. The numbers in bold are the first nodes of each block, and they correspond to the weights of those nodes of t that are put into set U. The structure tree in Figure 2 is built on this set U and is therefore a 5-coarse structure tree.

In the case of integer sequences, we showed that a coarse version of the input stream of size $\mathrm{O}(p/\epsilon)$ is sufficient to obtain a $(1+\epsilon)$-approximation. We will see that the more complicated structure of trees requires a coarse version of size $\Theta(p^2/\epsilon)$ for a $(1+\epsilon)$-approximation. Intuitively, this is due to the fact that, in the case of integer sequences, a partition is only adjacent to two other partitions. In the case of trees, a partition may be adjacent to $p-1$ other partitions. Hence, partition boundaries must be chosen more carefully, and a better resolution for our coarse object is required.

The main result of this Subsec., Lemma 10, states that, given a c-coarse structure tree $\mathrm{ST}(U)$, an optimal partitioning of $\mathrm{ST}(U)$ provides a partitioning of the original tree t such that the size of a partition increases at most by the additive term $2(p-1)c$. This is similar to Item 2 of Lemma 3 for integer sequences.

▶ **Lemma 10.** *Let U be such that $\mathrm{ST}(U)$ is a c-coarse structure tree of t. Consider an optimal partitioning S' of $\mathrm{ST}(U)$ into p parts. Consider the partitioning of t induced by S' and let B' denote the bottleneck value. Furthermore, let B^* denote the bottleneck value of an optimal partitioning of t. Then: $B' \leq B^* + 2(p-1)c$.*

Proof. Let S^* denote an optimal partitioning of t with bottleneck value B^*, let $t_1, \ldots, t_p$ denote the subtrees produced by S^*, and suppose that S^* is such that none of the subtrees t_i are empty (it is easy to see that there always is such a partitioning). Then, $S^* = \{\mathrm{rt}(t_1), \ldots, \mathrm{rt}(t_p)\}$. Denote by $U_i \subseteq U$ the subset of nodes of U that are also contained in t_i. Given S^*, we construct a partitioning $\tilde{S} = \{\tilde{s}_1, \ldots, \tilde{s}_p\}$ of $\mathrm{ST}(B)$ with bottleneck value $\tilde{B} \leq B^* + 2(p-1)c$. Since the optimal partitioning S' of $\mathrm{ST}(B)$ is at least as good, Fact 2 then implies the result.

Definition of $\tilde{S}$. For each partition t_i, we define a vertex $\tilde{s}_i$ of the partitioning $\tilde{S}$. Ideally, for every i, we would like to set $\tilde{s}_i = \mathrm{rt}(t_i)$, however, we can only do that if $\mathrm{rt}(t_i) \in \mathrm{ST}(U)$. If this is not the case, we select a nearby node contained in t_i that is also included in $ST(U)$, or, if the weight of t_i is not significant enough, we do not select any node, giving an empty partition. The definition of $\tilde{s}_i$ is as follows:

1. If $\mathrm{rt}(t_i) \in \mathrm{ST}(U)$: Let $\tilde{s}_i = \mathrm{rt}(t_i)$.
2. If $\mathrm{rt}(t_i) \notin \mathrm{ST}(U)$ and $\omega(t_i) \leq 2c$: Let $\tilde{s}_i = \perp$ (indicating that $\tilde{s}_i$ is unused).
3. If $\mathrm{rt}(t_i) \notin \mathrm{ST}(U)$ and $w(t_i) > 2c$: We define $\tilde{s}_i = \mathrm{lca}_t(B_i)$.
   For this to be a valid assignment, we will show in Lemma 12 that B_i is non-empty and in Lemma 13 that $\mathrm{lca}_t(B_i) \in \mathrm{ST}(B)$. We will prove in Lemma 14 that $\tilde{s}_i = \mathrm{lca}_t(B_i)$ is in a sense close to $\mathrm{rt}(t_i)$ by showing the following inequality on which we base our analysis: $\omega(\mathrm{stree}_t(\tilde{s}_i)) \geq \omega(\mathrm{stree}_t(\mathrm{rt}(t_i))) - 2c$.

Bounding $\tilde{B}$. We will now prove that the bottleneck value $\tilde{B}$ of the partitioning given by $\tilde{S}$ is at most $B^* + 2(p-1)c$. To see this, let $\tilde{t}_1, \ldots, \tilde{t}_p$ denote the subtrees induced by the partitioning $\tilde{S}$ such that $\mathrm{rt}(\tilde{t}_i) = \tilde{s}_i$ (if $\tilde{s}_i = \perp$ then let $\tilde{t}_i = \perp$). We will prove that $w(\tilde{t}_i) \leq \omega(t_i) + 2(p-1)c$, for every $\tilde{t}_i \neq \perp$, which then implies the result.

Now consider one partition $\tilde{t}_i$ with $\tilde{t}_i \neq \perp$. Let

$$J = \{j : \mathrm{rt}(t_j) \text{ is connected to a leaf of } t_i \text{ in } t\},$$

and let $J_\perp \subseteq J$ be those indices j such that $\tilde{s}_j = \perp$. Then:

$$\begin{aligned}
\omega(t_i) &= \omega(\mathrm{stree}_t(\mathrm{rt}(t_i))) - \sum_{j \in J} \omega(\mathrm{stree}_t(\mathrm{rt}(t_j))) \\
&\geq \omega(\mathrm{stree}_t(\tilde{s}_i))) - \sum_{j \in J_\perp} \omega(\mathrm{stree}_t(\mathrm{rt}(t_j))) - \sum_{j \in J \setminus J_\perp} \omega(\mathrm{stree}_t(\mathrm{rt}(t_j))), \qquad (1)
\end{aligned}$$

where we used $\omega(\mathrm{stree}_t(\mathrm{rt}(t_i))) \geq \omega(\mathrm{stree}_t(\tilde{s}_i)))$ since $\tilde{s}_i$ is contained in t_i. Then, due to Item 2 of the previous case distinction, we have $\omega(\mathrm{stree}_t(\mathrm{rt}(t_j))) \leq 2c$ for every $j \in J_\perp$, and, for every $j \in J \setminus J_\perp$, according to Items 1 and 3 of our case distinction, we have $\mathrm{stree}_t(\mathrm{rt}(t_j)) - \mathrm{stree}_t(\tilde{s}_j) \leq 2c$. Thus:

$$\begin{aligned}
\omega(t_i) &\geq \ldots \geq \omega(\mathrm{stree}_t(\tilde{s}_i))) - |J_\perp|2c - \sum_{j \in J \setminus J_\perp} (\omega(\mathrm{stree}_t(\tilde{s}_j)) + 2c) \\
&= \omega(\mathrm{stree}_t(\tilde{s}_i))) - \left(\sum_{j \in J \setminus J_\perp} \omega(\mathrm{stree}_t(\tilde{s}_j)) \right) - |J_\perp|2c - |J \setminus J_\perp|2c \\
&\geq \omega(\tilde{t}_i) - |J|2c \geq \omega(\tilde{t}_i) - (p-1)2c,
\end{aligned}$$

where we used $\omega(\tilde{t}_i) \leq \omega(\mathrm{stree}_t(\tilde{s}_i))) - \left(\sum_{j \in J \setminus J_\perp} \omega(\mathrm{stree}_t(\tilde{s}_j))\right)$ and $|J| \leq p-1$. The result follows. ◀

▶ **Corollary 11.** *Let U be such that $\mathrm{ST}(U)$ is a $S\epsilon/(2p^2)$-coarse structure tree where $S = \omega(t)$ for some $\epsilon > 0$. Then, an optimal partitioning of $\mathrm{ST}(U)$ into p parts provides a $(1+\epsilon)$-approximation to* Tree *on t.*

Proof. Let B' be the bottleneck value of an optimal partitioning P' of $\mathrm{ST}(U)$, and let B^* be the bottleneck value of an optimal partitioning of t. Then, by Fact 2, P' induces a partitioning of t with bottleneck value B'. Next, by Lemma 10, we have $B' \leq B^* + 2(p-1)c$, where $c = \epsilon/(2p^2)$. Furthermore, notice that $B^* \geq S/p$. Thus, we obtain the approximation ratio: $\frac{B'}{B} \leq \frac{B^* + 2(p-1)\cdot S\epsilon/(2p^2)}{B^*} < 1 + \frac{S\epsilon}{pB^*} \leq 1 + \epsilon$. ◀

Auxiliary Lemmas Used in the Proof of Lemma 10. We now present the technical lemmas that have been used in the proof of Lemma 10.

In Lemma 12, we show that for every subtree $\mathrm{stree}_t(v)$ of weight at least c, for some node v, at least one node of $\mathrm{stree}_t(v)$ is contained in every c-coarse structure tree.

▶ **Lemma 12 (*).** *Let U be such that $\mathrm{ST}(U)$ is a c-coarse structure tree of t. Let $v \in t$ be any node so that $\omega(\mathrm{stree}_t(v)) > c$. Then, $U \cap \mathrm{stree}_t(v) \neq \varnothing$.*

Lemma 13 is a simple structural property of trees.

▶ **Lemma 13.** *Let $U = \{u_1, u_2 \ldots\} \subseteq t$ be a subset of nodes of a tree of size at least two, ordered with respect to a depth-first-traversal. Then, there is an index $1 \leq i \leq |U| - 1$ such that $\mathrm{lca}(u_i, u_{i+1}) = \mathrm{lca}(U)$.*

Proof. Let $U_i = \{u_1, \dots, u_i\}$. We prove by induction on i that the statement is true for all sets U_i. Suppose that $i = 2$. Then trivially $\mathrm{lca}(u_1, u_2) = \mathrm{lca}(U_2)$. Now suppose that the statement is true for i and let x denote the $\mathrm{lca}(u_j, u_{j+1})$ for the value of j with $1 \leq j \leq i-1$ so that $x = \mathrm{lca}(U_i)$. If $u_{i+1} \in \mathrm{stree}_t(x)$ then clearly x is also the lowest-common-ancestor of U_{i+1}. Suppose now that $u_{i+1} \notin \mathrm{stree}_t(x)$. Let $y = \mathrm{lca}(u_i, u_{i+1})$. Then clearly $x \in \mathrm{stree}_t(y)$ and hence $U_i \subseteq \mathrm{stree}_t(y)$. Therefore, y is an ancestor of every node of U_{i+1}. It is also the lowest-common-ancestor of all nodes U_{i+1} as it is defined as $\mathrm{lca}(u_i, u_{i+1})$. ◀

Given a subtree $\mathrm{stree}_t(v)$ of weight at least c, for some node v, we show in Lemma 14 that the lowest-common-ancestor x of the nodes $U' = U \cap \mathrm{stree}_t(v)$ of a c-coarse structure tree $\mathrm{ST}(U)$ is in a sense close to v, i.e., $\omega(\mathrm{stree}_t(x)) + 2c \geq \omega(\mathrm{stree}_t(v))$.

▶ **Lemma 14 (*).** *Let U be so that $\mathrm{ST}(U)$ is a c-coarse structure tree of t. Let $v \in t$ be any node such that $\omega(\mathrm{stree}_t(v)) > c$, and let $U' = U \cap \mathrm{stree}_t(v)$. Furthermore, let $x = \mathrm{lca}(U')$. Then: $\omega(\mathrm{stree}_t(x)) + 2c \geq \omega(\mathrm{stree}_t(v))$.*

4.2 Computing Structure Trees

Algorithm. We use the first and the second pass to compute the IDs of the nodes $B \cup L$ with $B = \{b_1, \dots, b_k\}$ (we assume that they are ordered w.r.t. a depth-first-traversal of t) so that $\mathrm{ST}(B)$ is a $S\epsilon/(2p^2)$-coarse structure tree of t and $L = \{\mathrm{lca}(b_i, b_{i+1}) \,|\, 1 \leq i \leq k-1\}$. Then, in the third pass, we establish $\mathrm{ST}(B)$.

1st pass. Consider the subsequence of down-steps X_d of the input stream X and let X' denote the sequence of weights that constitute the second parameters of the tuples of X_d. Compute a $(\frac{S\epsilon}{2p^2})$-coarse version of X' by running the algorithm for partitioning integer sequences on X'[4]. Annotate every breakpoint of B created by the algorithm with the ID of the node that lead to its creation. This guarantees access to the IDs of the nodes B.

2nd pass. Concerning the nodes in L, we make use of the following observation: Let b_i, b_{i+1} be two nodes for which breakpoints are stored and $b_{i+1} \notin \mathrm{stree}_t(b_i)$. Let d_i be the minimal depth of a down-step between the down-step describing b_i and the down-step describing b_{i+1}. Then $x = \mathrm{lca}_t(b_i, b_{i+1})$ has depth $d_i - 1$. In the first pass, while simultaneously running the algorithm for partitioning integer sequences, we compute this depth (by keeping track of the minimal depth between any two nodes stored as first elements of two consecutive tuples in variable Y in Algorithm 1). Then, the down-step of node x, and hence the ID of x, can be identified in a second pass as the down-step at depth $d_i - 1$ that appears before the down-step of b_i, but is closest to the down-step b_i in the input stream.

3rd pass. Let $I = \{i_1, i_2, \dots\}$ be the IDs of the nodes $B \cup L$ and suppose that $i_j \leq i_{j+1}$ for every $1 \leq j < |I|$. We describe how to build the structure tree inductively. Suppose that it is correctly built up to the node with ID i_j, that is, it is the correct structure tree of the subtree of t induced by all nodes with ID at most i_j. The substream $X_j = (\text{'d'}, w_j), \dots, (\text{'d'}, w_{j+1})$, where w_j and w_{j+1} are the weights of the nodes $\mathrm{node}_t(i_j)$ and $\mathrm{node}_t(i_{j+1})$, respectively, allows us to determine the relationship between $\mathrm{node}_t(i_j)$ and $\mathrm{node}_t(i_{j+1})$. If $\mathrm{node}_t(i_{j+1})$

[4] Algorithm 1 computes a $(\frac{S\epsilon}{p})$-coarse version. In order to compute a $(\frac{S\epsilon}{2p^2})$-coarse version instead, CProbe has to be invoked with parameter $\lfloor \frac{S\epsilon}{2p^2} \rfloor$ in Line 8, and s should be set to $8\lceil \frac{p^2}{\epsilon} \rceil$ in Line 1.

is in the subtree of $\text{node}_t(i_j)$, then we add an edge from $\text{node}_t(i_j)$ to $\text{node}_t(i_{j+1})$ in $\text{ST}(B)$. Otherwise, we consider the minimum depth d_j of a down-step in the substream X_j. We deduce that $x = \text{lca}_t(\text{node}_t(i_j), \text{node}_t(i_{j+1}))$ is at depth $d_j - 1$. Node x has already been added to $\text{ST}(B)$, and, by its depth, we can identify it and connect it to $\text{node}_t(i_{j+1})$ in $\text{ST}(B)$. Concerning updating the weights ω', for every down-step $(\text{'d'}, w)$ in the stream, we add the weight w to the closest ancestor c in the current $\text{ST}(B)$ of the node that is described by the current down-step.

Post-processing. We can therefore establish $\text{ST}(B)$ in three passes. As a post-processing step, we use an optimal linear time partitioning algorithm by Frederickson [9] in order to partition $\text{ST}(B)$. From the resulting partitioning, according to Corollary 11, we deduce a $(1+\epsilon)$-approximation to the partitioning of t. All steps other from running the algorithm of Theorem 7 can be implemented with $\text{O}(1)$ update time.

Furthermore, it can be seen that the second and the third passes can be merged into a single pass (not explicitly described here), leading to a two-pass algorithm.

▶ **Theorem 15.** *There is a det. two-pass $(1+\epsilon)$-approximation streaming algorithm for* Tree *with space $\text{O}(\frac{p^2}{\epsilon}\log(mn))$, worst case update time $\text{O}(1)$, and post-processing time $\text{O}(p^2/\epsilon)$.*

5 Lower Bounds for Partitioning Integer Sequences

5.1 A Linear Space Lower Bound for Exact Algorithms

In this section, we show that any possibly randomized exact streaming algorithm for Part that performs one pass over the input requires $\Omega(n)$ space. We show this by a reduction from the Index problem in one-way two-party communication complexity.

▶ **Definition 16** (Index Problem)**.** Let $S = (S_1, \ldots, S_N)$ where $S \in \{0,1\}^N$, and let $I \in \{1, \ldots, N\}$. Alice is given S, Bob is given I. Alice sends message M to Bob and, upon reception, Bob outputs S_I.

We consider a version of Index where the index I is chosen from the set $\{\lceil N/2\rceil, \ldots, N\}$ uniformly at random. It is well-known [17] that the one-way randomized communication complexity of Index is $\Omega(N)$, and the modification in the input distribution restricting the index I to be chosen from the set $\{\lceil N/2\rceil, \ldots, N\}$ clearly changes the communication complexity only by a constant factor.

▶ **Lemma 17** (Hardness of the Index Problem)**.** *If S is chosen uniformly at random from $\{0,1\}^N$, I is chosen uniformly at random from the set $\{\lceil N/2\rceil, \ldots, N\}$, and the failure probability of the protocol is at most $1/3$, then $\mathbb{E}_S|M| = \Omega(N)$.*

Reduction. Given a streaming algorithm ALG that solves Part on a stream of length at most $3n$ using space s, we specify a protocol for an arbitrary instance (S, I) of Index with $|S| = n$, such that the message size is at most s.

Remember that Alice holds $S \in \{0,1\}^N$ and Bob holds $I \geq \lceil N/2\rceil$. Our protocol is the following: Alice generates the sequence $Y \in \{1,3\}^{2N}$ such that $Y_i = 2 \cdot S_{i/2} + 1$ for even i, and $Y_i = 4 - Y_{i+1}$ for odd i. Bob generates the sequence $Z = \underbrace{4 \ldots 4}_{2I-N-1}\ 2$.

Alice runs ALG on sequence Y with number of partitions $p = 2$. Once Y is entirely processed, she sends the resulting memory state of ALG to Bob. Bob continues running

ALG on Alice's final memory state and feeds the sequence Z into ALG. The message size of the protocol equals the space usage of ALG after processing Y. Note that ALG outputs the separator s_1 that separates the two partitions. If s_1 is even then Bob outputs 0, and if s_1 is odd then Bob outputs 1.

We prove that the above protocol is correct, which immediately yields the result.

▶ **Theorem 18.** *Any possibly randomized one-pass streaming algorithm for* Part *with error probability at most* $1/3$ *requires space* $\Omega(n)$.

Proof. First observe that $\sum_i Y_i + \sum_i Z_i = 4 \cdot N + (2I - N - 1) \cdot 4 + 2 = 8I - 2$. Let s_1^* denote the optimal split position. Suppose that a perfect balancing is achieved and the optimal bottleneck value is $4I - 1$. Since for all $i = 1, \ldots, N$ we have $Y_{2i-1} + Y_{2i} = 4$, this can only be achieved if s_1^* is even and $Y_{s_1^*} = 1$ which implies that $S_{s_1^*/2} = S_I = 1$. Suppose now that a perfect balancing cannot be achieved. This can only happen if s_1^* is odd and $Y_{s_1^*-1} = 3$ which implies that $S_{(s_1^*-1)/2} = S_I = 3$. Thus, the protocol is correct in both cases, and ALG can be used to solve Index. Lemma 17 implies the result. ◀

5.2 $\Omega(\frac{1}{\epsilon} \log n)$ Space Lower Bound for Approximation Algorithms

We prove now an $\Omega(\frac{1}{\epsilon} \log n)$ space lower bound for one-pass algorithms for Part that compute a $(1+\epsilon)$-approximation. We prove this lower bound in the one-way two-party communication setting for instances of Part with $m = 1$ and $p = 2$. Alice is given a sequence $Y \in \{0,1\}^n$ and Bob is given a sequence $Z \in \{0,1\}^n$, and they have to split the sequence $X = Y \circ Z$ into two parts. Alice sends a message to Bob, and, upon reception, Bob outputs the separator.

Input Distribution. Let t be an integer that is to be determined later. Alice's input and Bob's input are independent from each other and they are constructed as follows:

- **Alice's input** Y is a sequence of length n with $2(t-1)$ leading 1s, followed by an arbitrary sequence of length $n - 3t + 2$ with elements from $\{0, 11\}$ (11 is a pair of ones), where the number of 11s is exactly t. Denote by $\mathcal{Y}$ the set of all such sequences. Then Y is chosen uniformly at random from $\mathcal{Y}$. Clearly, the weight of Y is $4t - 2$, and $|\mathcal{Y}| = \binom{n-3t+2}{t}$.
- **Bob's input** Z is a sequence of length n with the first $4(i-1)$ elements 1, and the remaining elements 0, for some $i \in \{1, 2, \ldots, t\}$. Denote all such sequences as $\mathcal{Z}$. Then Z is chosen uniformly at random from $\mathcal{Z}$. Observe that the weight of Z varies from 0 to $4(t-1)$, and $|\mathcal{Z}| = t$.

Note that an optimal partitioning of any $Y \circ Z$ instance splits one of the 11s in the second part of Alice's input.

Example: Let $t = 2$, $n = 10$, and $p = 2$. Suppose that Alice holds $Y = 11\,00110110$. Bob's possible inputs are $Z_1 = 0000000000$ and $Z_2 = 1111000000$ of weight 0 and 4, respectively. The optimal partitioning of $Y \circ Z_1$ is $11\,001 \,|\, 10110\,0 \ldots 0$ and of $Y \circ Z_2$ is $11\,001101 \,|\, 10\,11110 \ldots 0$.

We give a lower bound on the communication complexity of any possibly randomized communication protocol that solves instances of $\mathcal{Y} \times \mathcal{Z}$ exactly.

▶ **Lemma 19.** *Any randomized one-way two-party protocol with error at most* $\delta > 0$ *that solves* Part *on instances of* $\mathcal{Y} \times \mathcal{Z}$ *has communication complexity at least* $\log\left(\frac{\binom{n-3t+2}{t}}{8\binom{t}{4\delta t} n^{4\delta t}}\right)$.

Proof. Let P be a randomized protocol as in the statement of the lemma. Then, by Yao's Lemma [28], there is a deterministic protocol Q with distributional error at most δ that has the same communication complexity. We prove a lower bound on the communication complexity of Q.

Denote by $M_1, \ldots, M_k$ the possible messages from Alice to Bob, and let $\mathcal{Y}_i \subseteq \mathcal{Y}$ denote the set of inputs that Alice maps to message M_i. Note that for a fixed input for Bob, the protocol Q outputs the same result for all inputs in $\mathcal{Y}_i$. Let $p_i = \Pr_{Y \leftarrow \mathcal{Y}_i, Z \leftarrow \mathcal{Z}}[Q \text{ errs on } (Y,Z)]$. Since the distributional error of the protocol is δ, or, in other words, $\Pr_{Y \leftarrow \mathcal{Y}, Z \leftarrow \mathcal{Z}}[Q \text{ errs on } (Y,Z)] \leq \delta$, we obtain $\frac{\sum_i p_i |\mathcal{Y}_i|}{|\mathcal{Y}|} \leq \delta$. Let $i \in \{1, \ldots, l\}$ be the indices for which $p_i \leq 2\delta$. Then by the Markov Inequality, $\sum_{i=1}^{l} |\mathcal{Y}_i| \geq \frac{1}{2}|\mathcal{Y}|$.

We bound $|\mathcal{Y}_i|$ from above for all $i \in \{1, \ldots, l\}$. First, note that for a particular input $Z \in \mathcal{Z}$, the output of Q on (Y,Z) is the same for all $Y \in \mathcal{Y}_i$. Denote by $\mathcal{Y}_i^j$ the subset of $\mathcal{Y}_i$ such that for each $Y^j \in \mathcal{Y}_i^j : \Pr_{Z \leftarrow \mathcal{Z}}[Q \text{ errs on } (Y^j, Z)] = \frac{j}{t}$, or, in other words, there are j inputs of Bob such that the protocol fails on Y^j, and, for the remaining $t-j$ inputs of Bob, the protocol succeeds. Consider the set $\mathcal{Y}_i^0$, i.e., for each $Y \in \mathcal{Y}_i^0$, the protocol succeeds on any input of Bob. This determines all positions of the pairs of 1s in Alice's input, and, therefore, there is only a single such element and we obtain $|\mathcal{Y}_i^0| \leq 1$. Similarly, we obtain $|\mathcal{Y}_i^j| \leq \binom{t}{j} n^j$, since the protocol errs on at most j inputs of Bob, therefore the position of $t-j$ pairs of 1s is fixed and only j pairs of 1s may differ (we allow them to have an arbitrary position in Y which is a very rough but sufficient estimate).

We apply the Markov Inequality again: For at least half of the elements of $\mathcal{Y}_i$, the protocol errs with probability at most 4δ. Therefore, $\frac{1}{2}|\mathcal{Y}_i| \leq \sum_{j \leq 4\delta t} |\mathcal{Y}_i^j| \leq \sum_{j \leq 4\delta t} \binom{t}{j} n^j \leq 2\binom{t}{4\delta t} n^{4\delta t}$, and thus $|\mathcal{Y}_i| \leq 4\binom{t}{4\delta t} n^{4\delta t}$. This then implies $l \geq \frac{|\mathcal{Y}|}{8\binom{t}{4\delta t} n^{4\delta t}} = \frac{\binom{n-3t+2}{t}}{8\binom{t}{4\delta t} n^{4\delta t}}$. Since the protocol sends at least l different messages, the communication complexity of the protocol is at least $\log(l)$, which implies the result. ◀

We make t small enough so that a solution to any instance of $\mathcal{Y} \times \mathcal{Z}$ that is a $(1+\epsilon)$-approximation actually solves the instance exactly. This idea leads to the following theorem:

▶ **Theorem 20.** *Any randomized one-way two-party communication protocol with error at most $\delta > 0$ (δ sufficiently small) that computes a $(1+\epsilon)$-approximation ($\frac{1}{\epsilon} = O(n^{1-\gamma})$ for any $\gamma > 0$) to* Part *on instances of $\mathcal{Y} \times \mathcal{Z}$ has communication complexity at least $\Omega\left(\frac{1}{\epsilon} \log n\right)$.*

Proof. We choose t small enough that a solution to any instance of $\mathcal{Y} \times \mathcal{Z}$ that is a $(1+\epsilon)$-approximation actually solves the instance exactly. Remark again that the weight of Y is $4t-2$ and the weight of Z is $4(i-1)$. Since the total weight is even, there is always a partitioning with weight $2t-1+2(i-1)$. Therefore, any partitioning that does not achieve an optimal balancing has an approximation factor of at least $\frac{2t-1+2(i-1)+1}{2t-1+2(i-1)}$, and we wish to choose t such that this approximation factor is worse than a $(1+\epsilon)$ approximation. Therefore, we have to choose t small enough such that for any $i \in \{1, 2, \ldots, t\}$: $\frac{1}{2t-1+2(i-1)} > \epsilon$, which implies that $t < \frac{1}{4\epsilon} + \frac{3}{4}$. We choose $t = \frac{1}{4\epsilon}$ and plug this value into the communication lower bound from Lemma 19. Using standard bounds on binomial coefficients:

$$\begin{aligned}
\Omega\left(\log\left(\frac{\binom{n-3t+2}{t}}{8\binom{t}{4\delta t} n^{4\delta t}}\right)\right) &= \Omega\left(\log\left(\frac{\left(4\epsilon(n-\frac{3}{4\epsilon}+2)\right)^{\frac{1}{4\epsilon}}}{8n^{\delta/\epsilon}\left(\frac{e}{4\delta}\right)^{\delta/\epsilon}}\right)\right) \\
&= \Omega\left(\frac{1}{4\epsilon}\log(4\epsilon n - 3 + 8\epsilon) - \frac{\delta}{\epsilon}\log(\frac{ne}{4\delta})\right) \\
&= \Omega\left(\frac{1}{4\epsilon}\log(4\epsilon n) - \frac{\delta}{\epsilon}\log(\frac{ne}{4\delta})\right) = \Omega(\left(\frac{1}{\epsilon}\log n\right),
\end{aligned}$$

for a sufficiently small but constant δ, and $\epsilon = \mathrm{O}(n^{1-\gamma})$ for any $\gamma > 0$. The result follows. ◀

6 Conclusion

In this paper, we initiated the study of the problems of partitioning integer sequences and partitioning trees in the streaming model. We showed that, for both problems, smaller versions of the input instances can be computed in a streaming fashion and still capture $(1+\epsilon)$-approximate partitionings of the original instances. For integer sequences, the small instances are of size $O(\frac{p}{\epsilon})$, and for trees, the small instances are of size $O(\frac{p^2}{\epsilon})$, both independent of the length of the input stream. Furthermore, for the problem of partitioning integer sequences, we provided space lower bounds obtained through communication complexity.

It remains to be investigated whether the sizes of the small instances for trees can be reduced to $O(\frac{p}{\epsilon})$. Furthermore, we conjecture that the number of passes of our algorithm for Tree can be reduced from two to one.

Acknowledgements. The author thanks László Kozma for many valuable ideas and helpful discussions, and an anonymous reviewer for improving the baseline algorithm stated in Fact 1.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: Sparsification, spanners, and subgraphs. In *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS'12, pages 5–14, New York, NY, USA, 2012. ACM.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In Prasad Raghavendra, Sofya Raskhodnikova, Klaus Jansen, and JoséD.P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 8096 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40328-6_1.

3 Konstantin Andreev and Harald Räcke. Balanced graph partitioning. In *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA'04, pages 120–124, New York, NY, USA, 2004. ACM. doi:10.1145/1007912.1007931.

4 Michael Bader. *Space-Filling Curves – An Introduction with Applications in Scientific Computing*. Texts in Computational Science and Engineering. Springer, 2013.

5 Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: Theory and algorithms. *Commun. ACM*, 56(8):87–94, August 2013.

6 S. H. Bokhari. Partitioning problems in parallel, pipeline, and distributed computing. *IEEE Trans. Comput.*, 37(1):48–57, January 1988. doi:10.1109/12.75137.

7 Vanessa Braganholo and Marta Mattoso. A survey on xml fragmentation. In *SIGMOD'14*, New York, NY, USA, 2014. ACM.

8 Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014*, volume 8635 of *Lecture Notes in Computer Science*, pages 275–286. Springer Berlin Heidelberg, 2014.

9 Greg N. Frederickson. Optimal algorithms for tree partitioning. In *SODA'91*, pages 168–177, Philadelphia, PA, USA, 1991. URL: http://dl.acm.org/citation.cfm?id=127787.127822.

10 Yijie Han, Bhagirath Narahari, and Hyeong-Ah Choi. Mapping a chain task to chained processors. *Inf. Process. Lett.*, 44(3):141–148, 1992. doi:10.1016/0020-0190(92)90054-Y.

11 Pierre Hansen and Keh-Wei Lih. Improved algorithms for partitioning problems in parallel, pipelined, and distributed computing. *IEEE Trans. Comput.*, 1992.

12 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570, 2014.

13 Sanjeev Khanna, S. Muthukrishnan, and Steven Skiena. Efficient array partitioning. In *ICALP*, volume 1256, pages 616–626. Springer Berlin Heidelberg, 1997. doi:10.1007/3-540-63165-8_216.

14 Jin Kim and Hyunchul Kang. A method of XML document fragmentation for reducing time of XML fragment stream query processing. *Computing and Informatics*, 31(3):639, 2012. URL: http://www.cai.sk/ojs/index.php/cai/article/view/1012.

15 Christian Konrad. Two-constraint domain decomposition with space filling curves. *Parallel Comput.*, 37(4-5):203–216, April 2011. doi:10.1016/j.parco.2011.03.002.

16 Christian Konrad and Frédéric Magniez. Validating xml documents in the streaming model with external memory. *ACM Trans. Database Syst.*, 38(4), 2013. doi:10.1145/2504590.

17 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

18 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization – preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond – Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, pages 129–161, 2012.

19 F. Magniez, C. Mathieu, and A. Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 2014.

20 Fredrik Manne and Bjørn Olstad. Efficient partitioning of sequences. *IEEE Trans. Comput.*, 44(11):1322–1326, November 1995. doi:10.1109/12.475128.

21 Fredrik Manne and Tor Sørevik. Optimal partitioning of sequences. *J. Algorithms*, 19(2):235–249, September 1995. doi:10.1006/jagm.1995.1035.

22 Serge Miguet and Jean-Marc Pierson. Heuristics for 1d rectilinear partitioning as a low cost and high quality answer to dynamic load balancing. In *HPCN Europe 1997*, pages 550–564, London, UK, UK, 1997. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645561.659355.

23 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS'10, pages 815–823, Washington, DC, USA, 2010. IEEE Computer Society.

24 Yehoshua Perl and Stephen R. Schach. Max-min tree partitioning. *J. ACM*, 28(1):5–15, January 1981. doi:10.1145/322234.322236.

25 Ali Pinar and Cevdet Aykanat. Fast optimal load balancing algorithms for 1d partitioning. *J. Parallel Distrib. Comput.*, 64(8):974–996, August 2004. doi:10.1016/j.jpdc.2004.05.003.

26 Stefan Schamberger and Jens-Michael Wierum. Partitioning finite element meshes using space-filling curves. *Future Gener. Comput. Syst.*, 21(5):759–766, May 2005. doi:10.1016/j.future.2004.05.018.

27 Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC'01, pages 183–192, New York, NY, USA, 2001. ACM.

28 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *FOCS'77*, pages 222–227, Washington, DC, USA, 1977. doi:10.1109/SFCS.1977.24.

29 Qirun Zhang, Michael R. Lyu, Hao Yuan, and Zhendong Su. Fast algorithms for dyck-cfl-reachability with applications to alias analysis. In *PLDI'13*, 2013. doi:10.1145/2491956.2462159.

Dynamic Graph Queries*

Pablo Muñoz[†1], Nils Vortmeier[2], and Thomas Zeume[‡3]

1 **Department of Computer Science, University of Chile, CIWS Center for Semantic Web Research, Santiago, Chile**
pmunoz@dcc.uchile.cl

2 **TU Dortmund University, Dortmund, Germany**
nils.vortmeier@tu-dortmund.de

2 **TU Dortmund University, Dortmund, Germany**
thomas.zeume@tu-dortmund.de

Abstract

Graph databases in many applications – semantic web, transport or biological networks among others – are not only large, but also frequently modified. Evaluating graph queries in this dynamic context is a challenging task, as those queries often combine first-order and navigational features.

Motivated by recent results on maintaining dynamic reachability, we study the dynamic evaluation of traditional query languages for graphs in the descriptive complexity framework. Our focus is on maintaining regular path queries, and extensions thereof, by first-order formulas. In particular we are interested in path queries defined by non-regular languages and in extended conjunctive regular path queries (which allow to compare labels of paths based on word relations). Further we study the closely related problems of maintaining distances in graphs and reachability in product graphs.

In this preliminary study we obtain upper bounds for those problems in restricted settings, such as undirected and acyclic graphs, or under insertions only, and negative results regarding quantifier-free update formulas. In addition we point out interesting directions for further research.

1998 ACM Subject Classification F.4.1. Mathematical Logic

Keywords and phrases Dynamic descriptive complexity, graph databases, graph products, reachability, path queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.14

1 Introduction

Graph databases are important in applications in which the topology of data is as important as the data itself. Intuitively, a graph database represents objects (by nodes), and relationships between those objects (often modeled by labeled edges – see [1] for a survey on graph database models). The last years have witnessed an increasing interest in graph databases, due to the uprise of applications that need to manage and query massive and highly-connected data, as for example the semantic web, social networks or biological networks. In most of these applications, databases are not only large, but also highly dynamic. Data is frequently

* Parts of this work are also included in the dissertation thesis of the third author [23].

† The author acknowledges the financial support by Conicyt PhD scholarship and Millennium Nucleus Center for Semantic Web Research under Grant NC120004.

‡ The author acknowledges the financial support by DFG grant SCHW 678/6-1.

inserted and deleted, and hence so is its network structure. The goal of this work is to explore how query languages for graph databases can be evaluated in this dynamic context.

Many query languages for graph databases combine traditional first-order features with *navigational* ones. Already basic languages (such as regular path queries, see e.g. [21, 2]) allow to test the existence of paths satisfying constraints on their labels (e.g. adherence to a regular expression in regular path queries). Computing the answers to this kind of queries on large, highly dynamic graphs is a big challenge. It is conceivable, though, for answers to a query before and after small modifications to be closely related. Thus a reasonable hope is to be able to update the answer to a query in a more efficient way than recomputing it from scratch after each modification. Even more so if we allow to store extra auxiliary data that might ease the updating task. To what extent this is possible, and in which precise conditions, is the subject of *dynamic computational complexity*.

Here we are interested in studying the dynamic complexity of query languages for graph databases from a *descriptive* approach. In the dynamic descriptive complexity setting, proposed independently by Dong, Su and Topor [9, 8] and by Patnaik and Immerman [16], a *dynamic program* maintains auxiliary relations with the intention to help answering a query over a (relational) database subject to small modifications (insertions or deletions of tuples). When a modification occurs, the query answers and every auxiliary relation are updated by first-order formulas (or, equivalently, by core SQL queries) evaluated over the current database and the available auxiliary data. Such programs benefit therefore from being both highly parallelizable (due to the close connection of first-order logic and small depth boolean circuits) and readily implementable in standard relational database engines. The class of queries maintainable by first-order update formulas is called DynFO.

Query languages for graphs have, so far, not been studied systematically in the dynamic descriptive complexity setting. Very likely the main reason is that until recently it was not even known whether reachability in directed graphs could be maintained by first-order update formulas. That this indeed is possible was shown in [6], with the immediate consequence that all fixed (conjunctions of) regular path queries can also be maintained. Thus regular path queries can be evaluated in a highly parallel fashion in dynamic graph databases.

Motivated by this result we study the dynamic maintainability of more expressive query languages.

▶ **Goal.** *Gain a better understanding of the limits of maintaining graph query languages in the dynamic context.*

Our focus is on regular path queries and extensions thereof – non-regular path queries and extended conjunctive regular path queries (short: ECRPQs).

Some previous work on non-regular path queries has been done. Weber and Schwentick exhibited a context-free path query (the Dyck language D_2) that can be maintained in DynFO on acyclic graphs [20]. Also, for the simple class of path-shaped graph databases, formal language results can be transferred. Already Patnaik and Immerman pointed out that regular languages can be maintained in DynFO [16]. Later, Gelade et al. systematically studied the dynamic complexity of formal languages [11]. They showed, among other results, that regular languages can be maintained by quantifier-free update formulas, and that all context-free languages can be maintained in DynFO.

The second extension of regular path queries to be studied here are extended conjunctive regular path queries. In previous work it has been noticed that conjunctions of regular path queries (CRPQs) fall short in expressive power for modern applications of graph databases [4]. A feature commonly demanded by these applications is the comparison of

labels of paths defined by CRPQs based on relations of words (e.g. prefix, length constraints, fixed edit-distance). ECRPQs have been introduced to fulfill this requirement [4], that is, they generalize CRPQs by allowing to test whether multiple labels of paths adhere to given regular relations. Two basic properties expressible by ECRPQs are whether two pairs of nodes are connected by paths of the same length and if so, whether also paths with the same label sequence exist. In general, maintaining the result of ECRPQs seems to be a difficult task. In this article we therefore explore the maintenance of ECRPQs in restricted settings.

Finally, there is also a close connection between the evaluation of graph queries and the reachability problem in unlabeled and labeled product graphs. We discuss this connection (see Section 2), and exploit it in several of our results.

Contributions. First we study path queries and show that

- all regular path queries can be maintained by quantifier-free formulas when only insertions are allowed,
- all context-free path queries can be maintained by first-order formulas on acyclic graphs, and
- there are non-context-free path queries maintainable by first-order formulas on undirected and acyclic graphs, as well as on general graphs under insertions only.

As a first step towards maintaining ECRPQs we explore for which graph classes the lengths of paths between nodes can be maintained. We exhibit dynamic programs for maintaining all distances for undirected and acyclic graphs, as well as for directed graphs when only insertions are allowed. It remains open, whether distances can be maintained in DynFO for general directed graphs, but we show that quantifier-free update formulas do not suffice.

The techniques used to maintain all distances can be used to maintain variants of ECRPQs in restricted settings. Denote the extension of a class of queries by linear constraints on the number of occurrences of symbols on paths by +LC. This extension was introduced and studied in [4]. We show that

- all CRPQ+LCs can be maintained by first-order formulas when only insertions are allowed, and
- all ECRPQ+LCs can be maintained by first-order formulas on acyclic graphs.

An immediate consequence of our results for distances is that reachability can be maintained in products of (unlabeled) graphs for those restrictions. By using the dynamic program for maintaining the rank of matrices from [6], we extend this result to more general graph products. Furthermore we show that pairs of nodes connected by paths with the same label sequence can be maintained in acyclic graphs using first-order update formulas.

Related work. The maintenance of problems has also been studied from an algorithmic point of view. A good starting point for readers interested in upper bounds for dynamic algorithms is [18, 7]; a good starting point for lower bound techniques is the survey by Miltersen on cell probe complexity [14]. The upper bounds for reachability obtained in [18, 7] immediately transfer to dynamic algorithmic evaluation of regular path queries (using the reduction exhibited in [6]).

Outline. The dynamic setting and the basic graph query languages are introduced in Section 2. There we also discuss the connection between query evaluation and reachability in product graphs. Section 3 contains the results on maintaining graph queries. Our results

for maintaining distances and ECRPQs are presented in Section 4. In Section 5 some of the results for maintaining graph queries are transferred to reachability in graph products, and we also provide results for reachability in generalized graph products. We conclude in Section 6. Due to the space limit we omit some proofs or give only proof sketches in the body of this paper. Complete proofs can be found in the full version [15].

2 Preliminaries

In this section we introduce the dynamic complexity framework as well as the graph query languages used in this article.

Dynamic complexity framework. In this work we use the dynamic complexity framework as introduced by Patnaik and Immerman [16]. The following introduction of the framework is borrowed from previous work [25].

Intuitively, the goal of a dynamic program is to keep the result of a given query $\mathcal{Q}$ up to date while the database to be queried (the *input database*) is subject to tuple insertions and deletions. To this end the dynamic program stores auxiliary relations (the *auxiliary database*) with the aim that one of those relations always (that is, after every possible sequence of modifications), stores the result of $\mathcal{Q}$ for the current input structure. Whenever a tuple is inserted into or deleted from the input structure, each auxiliary relation is updated by the dynamic program by evaluating a specified first-order formula.

We make this more precise now. A *dynamic instance* of a query $\mathcal{Q}$ is a pair $(\mathcal{D}, \alpha)$, where $\mathcal{D}$ is a database over some finite domain D and α is a sequence of modifications to $\mathcal{D}$. Here, a *modification* is either an insertion of a tuple over D into a relation of $\mathcal{D}$ or a deletion of a tuple from a relation of $\mathcal{D}$. The result of $\mathcal{Q}$ for $(\mathcal{D}, \alpha)$ is the relation that is obtained by first applying the modifications from α to $\mathcal{D}$ and then evaluating $\mathcal{Q}$ on the resulting database. We use the Greek letters α and β to denote modifications as well as modification sequences. The database resulting from applying a modification α to a database $\mathcal{D}$ is denoted by $\alpha(\mathcal{D})$. The result $\alpha(\mathcal{D})$ of applying a sequence of modifications $\alpha \stackrel{\text{def}}{=} \alpha_1 \dots \alpha_m$ to a database $\mathcal{D}$ is defined by $\alpha(\mathcal{D}) \stackrel{\text{def}}{=} \alpha_m(\dots(\alpha_1(\mathcal{D}))\dots)$.

Dynamic programs, to be defined next, consist of an initialization mechanism and an update program. The former yields, for every (input) database $\mathcal{D}$, an initial state with initial auxiliary data. The latter defines how the new state of the dynamic program is obtained from the current state when applying a modification.

A *dynamic schema* is a tuple $(\tau_{\text{in}}, \tau_{\text{aux}})$ where τ_{in} and τ_{aux} are the schemas of the input database and the auxiliary database, respectively. While τ_{in} may contain constants, we do not allow constants in τ_{aux} in the basic setting. We always let $\tau \stackrel{\text{def}}{=} \tau_{\text{in}} \cup \tau_{\text{aux}}$.

▶ **Definition 1** (Update program). An *update program* P over a dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a set of first-order formulas (called *update formulas* in the following) that contains, for every relation symbol R in τ_{aux} and every $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ with $S \in \tau_{\text{in}}$, an update formula $\phi^R_\delta(\bar{x}; \bar{y})$ over the schema τ where $\bar{x}$ and $\bar{y}$ have the same arity as S and R, respectively.

A *program state* $\mathcal{S}$ over dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a structure $(D, \mathcal{I}, \mathcal{A})$ where D is a finite domain, $\mathcal{I}$ is a database over the input schema (the *current database*) and $\mathcal{A}$ is a database over the auxiliary schema (the *auxiliary database*).

The semantics of update programs is as follows. Let P be an update program, $\mathcal{S} = (D, \mathcal{I}, \mathcal{A})$ be a program state and $\alpha = \delta(\bar{a})$ a modification where $\bar{a}$ is a tuple over D and $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ for some $S \in \tau_{\text{in}}$. If P is in state $\mathcal{S}$ then the application of α yields the

new state $\mathcal{P}_\alpha(\mathcal{S}) \stackrel{\text{def}}{=} (D, \alpha(\mathcal{I}), \mathcal{A}')$ where, in $\mathcal{A}'$, a relation symbol $R \in \tau_{\text{aux}}$ is interpreted by $\{\bar{b} \mid \mathcal{S} \models \phi^R_\delta(\bar{a}; \bar{b})\}$. The effect $P_\alpha(\mathcal{S})$ of applying a modification sequence $\alpha \stackrel{\text{def}}{=} \alpha_1 \dots \alpha_m$ to a state $\mathcal{S}$ is the state $P_{\alpha_m}(\dots(P_{\alpha_1}(\mathcal{S}))\dots)$.

▶ **Definition 2** (Dynamic program)**.** A *dynamic program* is a triple $(P, \textsc{Init}, Q)$, where

- P is an update program over some dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$,
- $\textsc{Init}$ is a mapping that maps τ_{in}-databases to τ_{aux}-databases, and
- $Q \in \tau_{\text{aux}}$ is a designated *query symbol*.

A dynamic program $\mathcal{P} = (P, \textsc{Init}, Q)$ *maintains* a query $\mathcal{Q}$ if, for every dynamic instance $(\mathcal{D}, \alpha)$, the query result $\mathcal{Q}(\alpha(\mathcal{D}))$ coincides with the content of Q in the state $\mathcal{S} = P_\alpha(\mathcal{S}_{\textsc{Init}}(\mathcal{D}))$ where $\mathcal{S}_{\textsc{Init}}(\mathcal{D})$ is the initial state for $\mathcal{D}$, that is, $\mathcal{S}_{\textsc{Init}}(\mathcal{D}) \stackrel{\text{def}}{=} (D, \mathcal{D}, \textsc{Init}(\mathcal{D}))$.

The following example due to [16] shows how the transitive closure of an acyclic graph subject to edge insertions and deletions can be maintained in this set-up. The basic technique of this example will be crucial in some of the later proofs.

▶ **Example 3.** Consider an acyclic graph G subject to edge insertions and deletions. In the following, our goal is to maintain the transitive closure of G using a dynamic program with first-order update formulas. It turns out that if the graph is guaranteed to remain acyclic, then it is sufficient to store the current transitive closure relation in an auxiliary relation T. We follow the argument from [16].

When an edge (u, v) is inserted into G the following very simple rule updates T: there is a path from x to y after inserting (u, v) if (1) there was already a path from x to y before the insertion, or (2) there were paths from x to u and from v to y before the insertion. This rule can be easily specified by a first-order update formula that defines the updated transitive closure relation[1]: $\phi^T_{\text{INS}_E}(u, v; x, y) \stackrel{\text{def}}{=} T(x, y) \vee \big(T(x, u) \wedge T(v, y)\big)$.

Deletions are slightly more involved. There is a path ρ from x to y after deleting an edge (u, v) if there was a path from x to y before the deletion and (1) there was no such path via (u, v), or (2) there is an edge (z, z') on ρ such that u can be reached from z but not from z'. If there is still a path ρ from x to y, such an edge (z, z') must exist, as otherwise u would be reachable from y, contradicting acyclicity. This rule can be described by a first-order formula:

$$\phi^T_{\text{DEL}_E}(u, v; x, y) \stackrel{\text{def}}{=} T(x, y) \wedge \Big(\big(\neg T(x, u) \vee \neg T(v, y)\big) \vee \exists z \exists z' \big(T(x, z) \wedge E(z, z') \wedge (z \neq u \vee z' \neq v) \wedge T(z', y) \wedge T(z, u) \wedge \neg T(z', u)\big)\Big)$$

◀

A word on the initial input and auxiliary databases is due. As default we use the original setting of Patnaik and Immerman, where the input database is empty at the beginning, and the auxiliary relations are initialized by first-order formulas evaluated on the initial input database. When we use a different initialization setting we state it explicitly. In the literature several other settings have been investigated and we refer to [25, 23] for a detailed discussion.

The class of queries that can be maintained by first-order update formulas in the setting of Patnaik and Immerman is called[2] DynFO. Restricting update formulas to be quantifier-free yields the class DynProp.

[1] For simplicity we use the same names for elements and variables.

[2] In [25, 24, 22] the class DynFO comes with an arbitrary initialization, yet there the focus is on lower bounds.

When showing that a particular query is in DynFO we often assume that *arithmetic* on the domain is available from initialization time, that is, we assume the presence of relations $\leq, +, \times$ that are interpreted as a linear order – allowing to identify elements with numbers –, addition and multiplication on the domain. From a DynFO program that relies on built-in arithmetic, a program without built-in arithmetic can be constructed for all queries studied here by using a technique from [6].

▶ **Proposition 4** ([6, Theorem 4]). *Every domain-independent query $\mathcal{Q}$ that can be maintained in* DynFO *with built-in arithmetic can also be maintained in* DynFO.

Here, a query is *domain-independent* if its result does not change when elements are added to the domain.

Constructing a DynFO program for a specific query $\mathcal{Q}$ can be a tedious task. Such a construction can often be simplified by reducing $\mathcal{Q}$ to a query $\mathcal{Q}'$ for which a dynamic program has already been obtained. Such a reduction needs to be consistent with first order logic and its use in this dynamic context. A suitable kind of reductions are *bounded first-order reductions*. Intuitively, a query $\mathcal{Q}$ reduces to a query $\mathcal{Q}'$ via a bounded first-order reduction if a modification of an instance of $\mathcal{Q}$ induces constantly many, first-order definable modifications in a instance of $\mathcal{Q}'$. Note that if $\mathcal{Q}$ can be reduced to $\mathcal{Q}'$ via a bounded first-order reduction, then first-order update formulas for a modification of an instance for $\mathcal{Q}$ can be obtained by composing the first-order update formulas for the corresponding (first-order definable) modications of the instance of $\mathcal{Q}'$. We refer to [16] and [12] for a detailed exposition to bounded first-order reductions.

In this article we study dynamic programs for queries on (labeled) graphs. For most of our dynamic programs the precise encoding of graphs is not important. If the input to a query is a single Σ-labeled graph $G = (V, E)$ then it can, for example, be encoded by binary relations E_σ that store all σ-labeled edges, for all $\sigma \in \Sigma$. Similarly for constantly many graphs. Some of our results are for input databases that contain more than constantly many graphs. Those can be encoded in higher arity relations in a straightforward way. For example, linearly many graphs can be stored in ternary relations E_σ containing a tuple (g, u, v) if graph g contains a σ-labeled edge (u, v).

Graph databases and query languages. We review basic definitions of graph databases in order to fix notations and introduce the query languages used in this work.

A *graph database* over an alphabet Σ is a finite Σ-labeled graph $G = (V, E)$ where V is a finite set of nodes and E is a set of labeled edges $(u, \sigma, v) \subseteq V \times \Sigma \times V$. Here σ is called the *label* of edge (u, σ, v). Given a Σ-labeled graph $G = (V, E)$ and a symbol $\sigma \in \Sigma$, we denote by G_σ the projection of G onto its σ-labeled edges, that is, the graph G_σ has the edge set $\{(u, v) \mid (u, \sigma, v) \in E\}$. We say that a Σ-labeled graph G is *acyclic* if the graph $\cup_{\sigma\in\Sigma} G_\sigma$ is acyclic, and *undirected*, if for each $\sigma \in \Sigma$ the graph G_σ is undirected.

A *path* ρ in G from v_0 to v_m is a sequence of edges $(v_0, \sigma_1, v_1), \ldots, (v_{m-1}, \sigma_m, v_m)$ of G, for some length $m \geq 0$. The *label* of ρ, denoted by $\lambda(\rho)$, is the word $\sigma_1 \cdots \sigma_m \in \Sigma^*$. Paths of length zero are labeled by the empty string ϵ. For a formal language $L \subseteq \Sigma^*$, we say that ρ is an L-path if $\lambda(\rho) \in L$.

The basic building block of many graph query languages are *regular path queries* (short: RPQs). An RPQ selects all pairs of nodes in a Σ-labeled graph that are connected by an L-path, for a a given regular language $L \subseteq \Sigma^*$. Here we are interested in two extensions of regular path queries. One of them are path queries defined by non-regular languages, namely context-free and non-context-free languages.

The second extension to be studied, *extended conjunctive regular path queries* (short: ECRPQs), allows to define multiple paths and to compare their labels based on relations on words. In the following we give a short introduction to ECRPQs and refer to [4] for a detailed study.

In ECRPQs, paths are compared by *regular relations*. A k-ary regular relation R over alphabet Σ is defined by a finite state automaton $\mathcal{A}$ that synchronously reads k words over $\Sigma \cup \bot$, with $\bot \notin \Sigma$. The $\bot$ symbol is a padding symbol that may only occur at the end of a word, and therefore allows for processing words of different length. More formally $\mathcal{A}$ reads words over the alphabet $(\Sigma \cup \bot)^k$, and a k-tuple of words is in R if its corresponding string over $(\Sigma \cup \bot)^k$ is accepted by $\mathcal{A}$.

An ECRPQ is of the form $\mathcal{Q}(\vec{z}) \longleftarrow \bigwedge_{1\leq i\leq m}(x_i, \pi_i, y_i), \bigwedge_{1\leq j\leq t} R_j(\vec{\omega}_j)$ where

- each R_j is a regular relation over Σ (specified by some finite state automaton),
- $\vec{x} = (x_1, ..., x_m)$, $\vec{y} = (y_1, ..., y_m)$ and $\vec{z}$ are tuples of node variables such that the variables in $\vec{z}$ occur in $\vec{x}$ or $\vec{y}$, and
- $\vec{\pi} = (\pi_1, ..., \pi_m)$ and $\vec{\omega}_1, ..., \vec{\omega}_t$ are distinct tuples of path variables such that all variables in each $\vec{\omega}_j$ occur in $\vec{\pi}$.

In general both node and path variables can occur in the head of an ECRPQ. Here we focus on ECRPQs with heads containing node variables only.

The semantics of ECRPQs is defined in a natural way. For an ECRPQ $\mathcal{Q}$ of the above form, a Σ-labeled graph $G = (V, E)$, and mappings ν from node variables to nodes and μ from path variables to paths, we write $(G, \nu, \mu) \models \mathcal{Q}$ if

- $\mu(\pi_i)$ is a path in G from $\nu(x_i)$ to $\nu(y_i)$ for $1 \leq i \leq m$, and
- the tuple $(\lambda(\mu(\pi_{j_1})), ..., \lambda(\mu(\pi_{j_k})))$ belongs to the relation R_j for each $\vec{\omega}_j = (\pi_{j_1}, ..., \pi_{j_k})$.

The result of $\mathcal{Q}$ evaluated on G is defined by $Q(G) \stackrel{\text{def}}{=} \{\nu(\vec{z}) : (G, \nu, \mu) \models \mathcal{Q}\}$.

Product Graphs and Graph Query Languages. There is a strong connection between the evaluation problem for many graph query languages and the reachability query for products of labeled graphs. For example, the evaluation of a regular path query L on a labeled graph G can be reduced to reachability in the product graph $\mathcal{A} \times G$ where $\mathcal{A}$ is a finite state automaton for L. Product graphs also help for the evaluation of fragments of ECRPQs as well. We will exploit this connection at several places and therefore present some basic properties of product graphs next.

The *product graph* $\prod_i G_i$ of m Σ-labeled graphs $G_i = (V_i, E_i)$, $1 \leq i \leq m$, has nodes $\prod_i V_i$ and an edge $(\vec{x}, \vec{y})$ between two nodes $\vec{x} = (x_1, ..., x_m)$ and $\vec{y} = (y_1, ..., y_m)$ if there is a symbol $\sigma \in \Sigma$ such that $(x_i, \sigma, y_i) \in E_i$ for each $1 \leq i \leq m$. The graphs G_i are called *factors* of the graph product. Graph products for unlabeled graphs are defined analogously. The following well known property characterizes reachability in (labeled) product graphs.

▶ **Fact 5.** *Let $(G_i)_{1\leq i\leq m}$ be graphs (Σ-labeled graphs) with $G_i = (V_i, E_i)$ and let $\vec{x} = (x_1, \ldots, x_m), \vec{y} = (y_1, \ldots, y_m)$ be two pairs of nodes of $\prod_i G_i$. Then $\vec{y}$ is reachable from $\vec{x}$ in $\prod_i G_i$ if and only if there are paths ρ_i from x_i to y_i in G_i with $|\rho_i| \leq |\prod_i V_i|$, for $i \in \{1, \ldots, m\}$, and $|\rho_i| = |\rho_j|$ ($\lambda(\rho_i) = \lambda(\rho_j)$ respectively) for all $i, j \in \{1, \ldots, m\}$.* ◀

The preceding fact can be used in the dynamic context as well, i.e. it is compatible with bounded first-order reductions. More precisely, reachability in products of unlabeled graphs can be inferred from all distances in the factors. We say that *all distances up to n^c*, for $c \in \mathbb{N}$, are computed by a dynamic program if, for a graph G with n nodes and arithmetic

on the domain[3], it maintains a relation D that contains all tuples (x, y, ℓ) such that there is a path from x to y of length ℓ, for $0 \leq \ell \leq n^c$.

▶ **Proposition 6.** *The following problems are equivalent under bounded first-order reductions with built-in arithmetic:*

1. *Maintaining all distances up to n^2.*
2. *Maintaining reachability in the product of two graphs (both of them subject to modifications).*
3. *Maintaining reachability in the product of two graphs, one of them a fixed path.*

A similar equivalence can be established for problems related to reachability in products of Σ-labeled graphs:

▶ **Proposition 7.** *The following problems are equivalent under bounded first-order reductions:*

1. *Maintaining the existence of equally labeled paths between two pairs of nodes.*
2. *Maintaining reachability in the product of two Σ-labeled graphs.*
3. *Maintaining reachability in the product of two Σ-labeled graphs, one of them undirected.*
4. *Maintaining the palindrome path query on Σ-labeled graphs.*

3 Dynamic Path Queries

Path queries, as mentioned in the introduction, have almost not been studied in dynamic complexity before. Until recently not even the simple query induced by the language $L(a^*)$ was known to be in DynFO. Yet as an immediate consequence of the dynamic first-order update program for reachability exhibited in [6], all fixed regular path queries (and, since DynFO is closed under conjunctions, also conjunctions of them) can be maintained by first-order update formulas.

In this section we continue the exploration of the dynamic maintainability of path queries. We show that under insertions quantifier-free update formulas are sufficient to maintain (fixed) regular path queries, and that more expressive path queries can be maintained for restricted classes of graphs and constrained modifications.

▶ **Theorem 8.** *When only insertions are allowed then every regular path query can be maintained by quantifier-free update formulas.*

We conjecture that quantifier-free update formulas do not suffice to maintain RPQs under both insertions and deletions. This would imply that reachability can be maintained without quantifiers which seems to be very unlikely. A first step towards verifying this conjecture was done in [25] where it was shown that reachability cannot be maintained with binary quantifier-free programs.

Proof sketch (of Theorem 8). Let L be a regular path query and let $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ be a DFA with $L = L(\mathcal{A})$. The dynamic program uses a binary relation symbol $R_{p,q}$ for every pair of states $(p, q) \in Q^2$, as well as a binary designated query symbol R. The idea is that for a labeled graph G, the relation $R_{p,q}$ contains all tuples $(x, y) \in V^2$ such that $\mathcal{A}$, for some labeled path ρ from x to y, can read ρ by starting in state p and ending in state q.

[3] We note that from the arithmetic on the domain, arithmetic upto n^c can be defined using first-order formulas.

The update formulas for the relations $R_{p,q}$ are slightly more involved than the formulas for maintaining reachability under insertions. This is because $\mathcal{A}$ might reach a state q from a state p only by reading a labeled path from x to y that contains one or more loops. The crucial observation is, however, that for deciding whether (x, y) is in $R_{p,q}$ it suffices to consider paths that contain the node x at most $|Q|$ times (as paths that contain x more than $|Q|$ times can be shortened). This can be checked by quantifier-free update formulas. ◀

Capturing non-regular path queries by first-order update formulas seems to be significantly harder than capturing CRPQs. We provide only some preliminary results for restricted classes of graphs and modifications.

When all distances for all pairs of nodes can be maintained for a restricted class of graphs, then also non-regular and even non-contextfree path queries can be maintained (e.g. the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$). Later in Theorem 11 and Theorem 12 we show that distances can be kept up-to-date on acyclic and undirected graphs, as well as on directed graphs under insertions. This implies the following result.

▶ **Theorem 9.**

1. *There is a non-context-free path query that can be maintained in* DynFO *on acyclic and undirected Σ-labeled graphs.*
2. *There is a non-context-free path query that can be maintained in* DynFO *when only insertions are allowed.*

On acyclic graphs, all context-free path queries can be maintained. It is known that context-free languages are in DynFO [11] and that the Dyck language with two types of parentheses can be maintained on acyclic graphs [20]. Generalizing the techniques used for those two results yields the following theorem.

▶ **Theorem 10.** *All context-free path queries can be maintained in* DynFO *on acyclic graphs.*

To prove Theorem 10, we fix a context-free language L and a grammar $\mathcal{G} = (V, \Sigma, S, P)$ for L. We assume, without loss of generality, that $\mathcal{G}$ is in Chomsky normal form, that is, it has only rules of the form $X \to YZ$ and $X \to \sigma$. Furthermore, if $\epsilon \in L$ then $S \to \epsilon \in P$ and no right-hand side of a rule contains S. We write $Z \Rightarrow^* w$ if $w \in (\Sigma \cup V)^*$ can be derived from $Z \in V$ using rules of $\mathcal{G}$.

The dynamic program maintaining L on acyclic graphs uses $(2k+2)$-ary relation symbols $R_{X \to Y_1,\ldots,Y_k}$, for $k \in \{1, 2, 3\}$. The intention is that for an input graph database G, the relation $R_{X \to Y_1,\ldots,Y_k}$ contains a tuple $(x_1, y_1, \ldots, x_{k+1}, y_{k+1})$ if and only if there are strings $s_1, \ldots, s_{k+1} \in \Sigma^*$ such that $X \Rightarrow^* s_1 Y_1 s_2 \cdots s_k Y_k s_{k+1}$ and there is an s_i-path ρ_i from x_i to y_i in G. The paths ρ_i are called *witnesses* for $(x_1, y_1, \ldots, x_{k+1}, y_{k+1}) \in R_{X \to Y_1,\ldots,Y_k}$.

In a first step we prove that every relation $R_{X \to Y_1,\ldots,Y_k}$ is first-order definable from the relations $R_{X \to Y}$, so it actually suffices to only maintain these relations. This proof can be found in the full version of this paper.

Proof idea (of Theorem 10. Let L be an arbitrary context-free language and let $\mathcal{G} = (V, \Sigma, S, P)$ be a grammar for L in Chomsky normal form. We provide a DynFO-program $\mathcal{P}$ with designated binary query symbol Q that maintains L on acyclic graphs. The input schema is $\{E_\sigma \mid \sigma \in \Sigma\}$ and the auxiliary schema is $\tau_{\text{aux}} = \{R_{X \to Y} \mid X, Y \in V\} \cup \{T\}$. The intention of the auxiliary relation symbols $R_{X \to Y}$ has already been explained above; the relation symbol T shall store the transitive closure of the input graph (where the input graph is the union of all E_σ).

As already stated above, the query relation Q is first-order definable from the relations $R_{X\to Y}$, as a tuple (x, y) is in the query relation if and only if $x = y$ and $\epsilon \in L$, or there is a τ-labeled edge (z_1, z_2) such that $(x, z_1, z_2, y) \in R_{S\to U}$ for some $U \in V$ with $U \to \tau$.

It remains to present update formulas for each $R_{X\to Y}$. After inserting a σ-edge (u, v), a tuple (x_1, y_1, x_2, y_2) is contained in $R_{X\to Y}$ if there are two witness paths ρ_1 and ρ_2 such that (1) ρ_1 and ρ_2 have already been witnesses before the insertion, or (2) only ρ_1 uses the new σ-edge, or (3) only ρ_2 uses the new σ-edge, or (4) both ρ_1 and ρ_2 use the new σ-edge. In case (2) the path ρ_1 can be split into a path from x_1 to u, the edge (u, v) and a path from v to y_1. Similarly in the other cases and for ρ_2. This can be expressed using the first-order formulas defining $R_{X\to Y_1,\dots,Y_k}$.

After deleting a σ-edge (u, v) a tuple (x_1, y_1, x_2, y_2) is in $R_{X\to Y}$ if it still has witness paths ρ_1 and ρ_2 from x_1 to y_1 and from x_2 to y_2, respectively. The update formula for $R_{X\to Y}$ verifies that such witness paths exist. Therefore, similar to Example 3, the formula distinguishes for each $i \in \{1, 2\}$ whether (1) there was no path from x_i to y_i via (u, v) before deleting the σ-edge (u, v), or (2) there was a path from x_i to y_i via (u, v).

In case (1) all paths present from x_i to y_i before the deletion of the σ-edge (u, v) are also present after the deletion. In particular the set of possible witnesses ρ_i remains the same. For case (2), the update formula has to check that there is still a witness path ρ_i. Such a path ρ_i has the options (a) to still use the edge (u, v) but for a $\tau \neq \sigma$, and (b) to not use the edge (u, v) at all.

Whether some witness path uses (u, v) can be checked using the relation T. Existence of alternative witness paths can be verified similar to Example 3 with the relations T and $R_{X\to Y_1,\dots,Y_k}$. The complete update formulas can be found in the full version. ◀

4 Dynamic Extended Conjunctive Regular Path Queries

In this section we explore the maintainability of ECRPQs. In contrast to path queries, ECRPQs allow for testing properties of tuples of paths between pairs of nodes. Comparing the length of two paths is one of the simplest such properties and is therefore studied first. Afterwards we extend some of the techniques developed for maintaining the lengths of paths to ECRPQs.

4.1 Maintaining Distances

Maintaining all distances in arbitrary graphs is one of the big challenges of dynamic complexity. Recall that for maintaining all distances up to n^c a dynamic program has to update, for a graph G, a relation D that contains all tuples (x, y, ℓ) such that there is a path from x to y of length ℓ in G, for $0 \leq \ell \leq n^c$.

The recent dynamic algorithm for maintaining reachability (see [6]) does, unfortunately, not offer hints at how to maintain distances. A dynamic upper bound for distances is provided by Hesse's DynTC^0-program for reachability [13]. The program actually maintains the number of different paths of length ℓ between every pair of nodes, for any length ℓ up to the size of the graph, and thus all distances for all pairs of nodes. The program can be easily modified to compute all distances up to fixed polynomials.

Here we present preliminary results for maintaining all distances with first-order formulas for restricted modifications as well as for restricted classes of graphs. Furthermore we show that distances cannot be maintained with quantifier-free update formulas.

The *shortest* distance between every pair of nodes can be easily maintained in DynFO when edges can only be inserted; basically because shortest paths do not contain loops.

Maintaining *all* distances for all pairs of nodes under insertions requires some work.

▶ **Theorem 11.** *All distances up to $p(n)$ can be maintained in* DynFO *under insertions for every fixed polynomial $p(n)$.*

Proof. We describe how to maintain distances up to n; the generalization to distances up to $p(n)$ is straightforward and sketched at the end of the proof. The idea is to maintain a 4-ary relation A that contains a tuple (x, y, t, ℓ) if there are t (not necessarily distinct) paths from x to y such that the sum of their lengths is ℓ.

There is a path of length ℓ from node x to node y if and only if $(x, y, 1, \ell)$ holds. For maintaining this information, we need the full relation: a path from x to y can use a newly inserted edge (u, v) several times if cycles are present. Also, the path can use an arbitrary combination of cycles including that edge, and each cycle can be used arbitrarily often.

When inserting an edge (u, v) the updated relation A is defined by the following formula:

$$\phi^A_{\mathrm{INS}_E}(u, v; x, y, t, \ell) \stackrel{\text{def}}{=} \exists t_- \exists t_+ \exists t_{\circlearrowleft} \exists \ell_- \exists \ell_{+_1} \exists \ell_{+_2} \exists \ell_{\circlearrowleft}$$
$$\Big(A(x, y, t_-, \ell_-) \wedge A(x, u, t_+, \ell_{+_1}) \wedge A(v, y, t_+, \ell_{+_2}) \wedge A(v, u, t_{\circlearrowleft}, \ell_{\circlearrowleft})$$
$$\wedge (t_+ = 0 \to t_{\circlearrowleft} = 0) \wedge t_- + t_+ = t \wedge \ell_- + \ell_{+_1} + \ell_{+_2} + \ell_{\circlearrowleft} + t_+ + t_{\circlearrowleft} = \ell \Big)$$

If there are t paths with total length ℓ from x to y after the edge (u, v) is inserted, these paths can be divided into t_- paths that do not use the new edge (u, v), with a total length of ℓ_-, and t_+ paths that use the edge (u, v). Each one of these t_+ paths is composed of (i) one path from x to u that does not use (u, v), (ii) the edge (u, v), (iii) possibly some cycles from v back to v created by combining an old path from v to u and the new edge (u, v), and (iv) one path from v to y that does not use (u, v).

Without considering the cycles in v that use (u, v), in total there are t_+ paths from x to u (with total length ℓ_{+_1}), t_+ paths from v to y (with total length ℓ_{+_2}) and t_+ times the new edge (u, v). So these paths have total length $\ell_{+_1} + \ell_{+_2} + t_+$. Additionally, let $t_{\circlearrowleft}$ be the number of times the edge (u, v) is used in cycles from v to v in all t_+ paths together. These cycles can be obtained from $t_{\circlearrowleft}$ paths from v to u of total length $l_{\circlearrowleft}$ and $t_{\circlearrowleft}$ times the new edge (u, v). So in total, the t_+ paths have a total length of $\ell_{+_1} + \ell_{+_2} + t_+ + \ell_{\circlearrowleft} + t_{\circlearrowleft}$.

For maintaining distances upto $p(n)$, numbers of this magnitude are encoded by tuples of elements. Arithmetic upto $p(n)$ can be easily defined in a first-order fashion from the built-in arithmetic upto n. The above construction then translates in a straightforward way. ◀

Next we show that all distances for all pairs of nodes in undirected and acyclic graphs can be updated using first-order update formulas. For undirected graphs this slightly extends a result by Grädel and Siebertz [12] that the shortest distance can be maintained for undirected paths. For acyclic graphs the maintenance of all distances is a straight-forward extension of the dynamic program for maintaining reachability shown in Example 3.

▶ **Theorem 12.** *All distances up to $p(n)$ can be maintained in* DynFO *for every fixed polynomial $p(n)$ for* (a) *undirected graphs, and* (b) *acyclic graphs.*

The proof can be found in the full version.

In the rest of this subsection we discuss why distance information cannot be maintained by quantifier-free update formulas. So far the goal, when maintaining distances, was to store tuples (a, b, ℓ) in some relation if there is a path from a to b of length ℓ, where the length ℓ referred to the built-in arithmetic. It can be easily seen that maintaining distances in

this fashion is not possible with quantifier-free formulas (basically because a quantifier-free formula only has access to the numbers represented by the modified nodes).

Another way of maintaining distance information is to store a 4-relation that contains a tuple (a_1, a_2, b_1, b_2) if and only if there are paths from a_1 to a_2 and from b_1 to b_2 of equal length. We show that this relation cannot be maintained by quantifier-free programs.

Denote by Equal-Length-Paths the query on (unlabeled) graphs that selects all tuples (a_1, a_2, b_1, b_2) such that there are paths from a_1 to a_2 and from b_1 to b_2 of equal length.

▶ **Theorem 13.** *The query* Equal-Length-Paths *cannot be maintained by quantifier-free update formulas, even when the auxiliary relations can be initialized arbitrarily. In particular, ECRPQs and reachability in product graphs cannot be maintained in this setting either.*

Intuitively this is not very surprising. It is well known that non-regular languages and therefore, in particular, the language $\{a^n b^n \mid n \in \mathbb{N}\}$ cannot be maintained by a quantifier-free program [11]. Thus maintaining whether two isolated paths have the same length should not be possible either. Technical issues arise from the fact that the query Equal-Length-Paths is over graphs, not strings. Yet the techniques used for proving lower bounds for languages can be adapted, see the full paper for details.

4.2 Maintaining ECRPQs

Here we study the maintenance of ECRPQs and provide results in restricted settings. First we show that answers to an ECRPQ can be maintained in DynFO on acyclic graphs. Even more, answers to the following extension of ECRPQs introduced in [4] can still be maintained. An ECRPQ with *linear constraints on the number of occurrences of symbols* on paths over an alphabet $\Sigma = \{\sigma_1, ..., \sigma_k\}$ is of the form

$$\mathcal{Q}(\vec{z}) \longleftarrow \bigwedge_{1 \le i \le m} (x_i, \pi_i, y_i),\ \bigwedge_{1 \le j \le t} R_j(\vec{\omega}_j),\ A\vec{\ell} \ge \vec{b}$$

where $A \in \mathbb{Z}^{h \times (km)}$ for some $h \in \mathbb{N}$, $\vec{b} \in \mathbb{Z}^h$, and $\vec{\ell} = (\ell_{1,1}, ...\ell_{1,k}, ..., \ell_{m,1}, ..., \ell_{m,k})$. The semantics extends the semantics of ECRPQs as follows: for each $1 \le i \le m$ and $1 \le j \le k$, the variable $\ell_{i,j}$ is interpreted as the number of occurrences of the symbol σ_j in the path π_i. The last clause of the query $\mathcal{Q}$ is true if $A\vec{\ell} \ge \vec{b}$ under this interpretation.

▶ **Theorem 14.** *Every ECRPQ with linear constraints on the number of occurrences of symbols is maintainable in* DynFO *on acyclic graphs.*

Proof. Let $\Sigma = \{\sigma_1, ..., \sigma_k\}$. We show how to maintain the answer of an ECRPQ $\mathcal{Q}$ with linear constraints with only one regular relation R on an acyclic Σ-labeled graph $G = (V, E)$. Thus $\mathcal{Q}$ is of the form:

$$\mathcal{Q}(\vec{z}) \longleftarrow \bigwedge_{1 \le i \le m} (x_i, \pi_i, y_i), R(\pi_1, \ldots, \pi_m), A\vec{\ell} \ge \vec{b}$$

An arbitrary ECRPQ with linear constraints can be rewritten in this form by using closure properties of regular relations.

In a first step we reduce this problem to a structurally simpler one: the problem of maintaining $\mathcal{Q}$ on a Σ-labeled graph consisting of m disjoint acyclic graphs $G_1, \ldots, G_m$, restricted in such a way that solutions may only map the variables x_i, y_i to nodes in G_i, for each $1 \le i \le m$. The simple reduction from the original problem copies the queried graph m times. As m is a constant, this is a bounded first order reduction.

Let $\mathcal{A} = (Q, (\Sigma \cup \bot)^m, \delta, s, F)$ be a finite automaton with padding symbol $\bot \notin \Sigma$ that recognizes the m-ary regular relation R. The idea is to maintain $(2m + km)$-ary auxiliary relations $R_{p,q}$ for all $p, q \in Q$ intended to store a tuple $(\vec{x}, \vec{y}, \vec{\ell}_1, \ldots, \vec{\ell}_m)$ with $\vec{x} = (x_1, \ldots, x_m), \vec{y} = (y_1, \ldots, y_m)$ and $\vec{\ell}_i = (\ell_{i,1}, \ldots, \ell_{i,k})$ if and only if the state q is reachable from the state p in $\mathcal{A}$ by reading a tuple of words $(\lambda(\rho_1), \ldots, \lambda(\rho_m))$, where for each $1 \leq i \leq m$, ρ_i is a path in G_i from x_i to y_i, and $\ell_{i,1}, \ldots, \ell_{i,k}$ are the number of occurrences of the symbols $\sigma_1, \ldots, \sigma_k$ in the label sequence of ρ_i.

We show how to express the query relation Q by these relations. To this end observe that the (fixed) linear inequality system $A\vec{\ell} \geq \vec{b}$ can be defined by a $(m \times k)$-ary first-order formula $\psi_{A,\vec{b}}(\vec{\ell}_1, \ldots, \vec{\ell}_m)$ that uses the built-in arithmetic.

The query relation Q is then defined by the following formula:

$$\varphi(\vec{z}) \stackrel{\text{def}}{=} \exists \vec{v}\, \exists \vec{\ell}_1 \cdots \exists \vec{\ell}_m \bigvee_{f \in F} R_{s,f}(\vec{x}, \vec{y}, \vec{\ell}_1, \ldots, \vec{\ell}_m) \wedge \psi_{A,\vec{b}}(\vec{\ell}_1, \ldots, \vec{\ell}_m)$$

Here the existentially quantified variables $\vec{v}$ correspond to variables of $\mathcal{Q}$ that do not occur in the head of the query, and all x_i and y_i occur in either $\vec{z}$ or $\vec{v}$.

The update formulas for the relations $R_{p,q}$ are similar in spirit to those for reachability in acyclic graphs used in Example 3. They are described in detail in the full paper. ◀

It remains open whether the answer relation of ECRPQs can be maintained on general graphs, even when only insertions are allowed. Yet when the rational relations are restricted to be unary, the ECRPQs can be maintained under insertions. More formally, a *CRPQ with linear constraints on the number of occurrences of symbols* over $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is of the form

$$\mathcal{Q}(\vec{z}) \longleftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i),\ \bigwedge_{1 \leq j \leq m} L_j(\pi_j),\ A\vec{\ell} \geq \vec{b}$$

where L_j is a unary rational relation (that is, a regular language), and A, $\vec{b}$ and $\vec{\ell}$ are as in the definition of ECRPQs with linear constraints.

▶ **Theorem 15.** *Every CRPQ with linear constraints on the number of occurrences of symbols is maintainable in* DynFO *under insertions.*

Proof. Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ and $\mathcal{Q}$ be a CRPQ over Σ with linear constraints on the number of occurrences of symbols as above. Further let $\mathcal{A}_j = (Q_j, \Sigma, \delta_j, s_j, F_j)$, $1 \leq j \leq m$, be finite state automata for the regular languages L_j occurring in $\mathcal{Q}$.

We exhibit a DynFO-program with built-in arithmetic for maintaining Q on general graphs under insertions. The necessity for built-in arithmetic can be removed by Proposition 4.

The idea is similar to the proof of the previous Theorem 14. We maintain $(k+2)$-ary auxiliary relations $R^j_{p,q}$ for each $j \in \{1, \ldots, m\}$ and all $p, q \in Q_j$ with the intention that $R^j_{p,q}$ stores a tuple $(x, y, \ell_1, \ldots, \ell_k)$ if and only if the state q is reachable from state p in the automaton $\mathcal{A}_j$ by reading the label of a path ρ between x and y in G such that $\ell_1, \ldots, \ell_k$ are the number of occurrences of $\sigma_1, \ldots, \sigma_k$ in ρ.

Before sketching how to maintain the relations $R^j_{p,q}$, we show how they can be used to express the answer of $\mathcal{Q}$. As in the proof of Theorem 14 the (fixed) linear inequality system $A\vec{\ell} \geq \vec{b}$ can be defined by a $(m \times k)$-ary first-order formula $\psi_{A,\vec{b}}(\ell_{1,1}, \ldots, \ell_{m,k})$ that uses the built-in arithmetic. Then a tuple $\vec{u}$ of nodes in G is in the answer of $\mathcal{Q}$ if and only if the following formula holds:

$$\varphi(\vec{z}) \stackrel{\text{def}}{=} \exists \vec{v}\, \exists \ell_{1,1}, \ldots, \ell_{m,k}, \bigwedge_{1 \leq j \leq m} \left(\bigvee_{f \in F_j} R^j_{s_j,f}(x_j, y_j, \ell_{j,1}, \ldots, \ell_{j,k}) \right) \wedge \psi_{A,\vec{b}}(\ell_{1,1}, \ldots, \ell_{m,k})\,.$$

Here the existentially quantified variables $\vec{v}$ correspond to variables of $\mathcal{Q}$ that do not occur in the head of the query, and all x_j and y_j occur in either $\vec{v}$ or $\vec{z}$.

A small technical issue arises from the fact that it is not obvious why the length of paths $\rho_1, \ldots, \rho_m$ witnessing that a tuple of nodes $\vec{u}$ is in the answer of $\mathcal{Q}$ is polynomially bounded. This, however, is necessary for being able to quantify the length $\ell_{1,1}, \ldots, \ell_{m,k}$ and to use the built-in arithmetic for computations. Fortunately the length of (shortest) witness paths can be bounded by a fixed polynomial in the size of the active domain. This has been shown even for ECRPQs with such linear constraints in [4, Lemma 8.6].

Now we show how to maintain the relations $R^j_{p,q}$. The following notion is useful. A relation R stores the *Parikh distances* of a Σ-labeled graph if it contains a tuple $(x, y, \ell_1, \ldots \ell_k)$ if and only if there is a path ρ between x and y such that its label $\lambda(\rho_i)$ contains ℓ_i occurrences of the symbol σ_i for each $1 \leq i \leq m$. We observe that the relations $R^j_{p,q}$ can be defined from the Parikh distance relations of the product graphs $G \times A_j$. Since the automata A_j are fixed, a modification of G yields a bounded number of first-order definable modifications to $G \times A_j$.

Thus in order to maintain $R^j_{p,q}$, it suffices to be able to maintain the Parikh distance relation of a Σ-labeled graph under insertions. However, the dynamic program for maintaining distances under insertions from Theorem 11 can be easily generalized to maintain Parikh distances. ◀

We remark that already boolean ECRPQs cannot be maintained under insertions in DynProp due to lower bounds for non-regular languages [11], and boolean CRPQs with $k+2$ existentially quantified node variables cannot be maintained in DynProp with k-ary relations due to a lower bound for the k-clique query [24].

5 Maintaining Reachability in Product Graphs

In this final section we study the reachability query for product graphs. In addition to its importance for the evaluation of fixed graph queries, reachability in graph products can be used to maintain the result of regular path queries in combined complexity (i.e., when the query is subject to modifications as well). Furthermore it is relevant in model checking, where subsystems correspond to factors in product graphs (see, e.g., [3]).

The results for maintaining all distances obtained in the previous section immediately transfer to reachability in simple graph products (see the discussion at the end of Section 2). A small technical obstacle arises from the fact that the reachability query does not come with built-in arithmetic, while the distance query studied so far does. However, this is not a problem due to Proposition 4.

▶ **Theorem 16.** *Let $\mathcal{G}$ be a class of graphs and $m \in \mathbb{N}$. If all distances up to n^m on $\mathcal{G}$ can be maintained in* DynFO *with built-in arithmetic, then reachability in the product of m $\mathcal{G}$-graphs is maintainable in* DynFO *(without built-in arithmetic).*

Shortest paths in products of acyclic and undirected graphs are of length at most n and n^2, respectively. For these two classes of graphs, reachability can therefore be maintained in products of polynomially many factors using the program for all distances. More precisely, this is doable for reachability between two specified nodes $\vec{s}$ and $\vec{t}$ as opposed to all pairs of nodes (as there are exponentially many nodes in such product graphs).

For directed graphs, shortest paths in products of polynomially many graphs can be of exponential length. For this reason, the approach to maintain reachability in such products via distances fails. Even more, it is unlikely that there is a DynFO-program for this problem: it could be used to decide reachability in the product of polynomially many graphs in PTime,

which is NP-hard. This follows from a reduction from emptiness of intersections of unary regular expressions which is known to be NP-hard [10].

▶ **Corollary 17.** *Reachability can be maintained in* DynFO *in the product of*
1. *polynomially many undirected graphs,*
2. *polynomially many acyclic graphs, and*
3. *a constant number of directed graphs under insertions.*

This follows immediately from Theorem 16, Theorem 12 and Theorem 11. Reachability in products of an undirected and an acyclic graph and similar constellations can, of course, also be maintained.

For labeled graph products, the following corollary follows immediately from the proof of Theorem 14.

▶ **Corollary 18.** *Reachability in products of constantly many acyclic Σ-labeled graphs can be maintained in* DynFO*.*

In the following we generalize Corollary 17 to a broader class of graph products. In the product graphs considered so far, there is an edge from a node $(x_1, \dots, x_m)$ to a node $(y_1, \dots, y_m)$ if there is an edge (x_i, y_i) in every factor G_i. This can be seen as a completely synchronized traversal through the given graphs. The graph products to be introduced next allow for more flexible, partially synchronized traversals.

Let $(G_i)_{1 \le i \le m}$ be a sequence of graphs with $G_i \stackrel{\text{def}}{=} (V_i, E_i)$, and let $A \stackrel{\text{def}}{=} (\vec{a}_1, \dots, \vec{a}_k)$ be a list of tuples from $\{0,1\}^m$, called *transition rules*. We often identify A with the matrix that has the tuples $\vec{a}_i$ as columns. The *generalized graph product of $(G_i)_i$ with respect to A*, denoted $\prod_i^A G_i$, has nodes $V_1 \times \cdots \times V_m$ and edges $(\vec{x}, \vec{y})$ defined by the first-order formula

$$\bigvee_{\substack{\vec{a} \in A \\ \vec{a}=(a_1,\dots,a_m)}} \bigwedge_{a_i=0} x_i = y_i \;\wedge\; \bigwedge_{a_i=1} E_i(x_i, y_i)\,.$$

For example, the usual product of two graphs is defined by the transition rule $\{(1,1)\}$, and the so called *cartesian product* is defined by the rules $\{(1,0),(0,1)\}$. We remark that generalized graph products have also been called *non-complete extended p-sums* (see [19]).

▶ **Theorem 19.** *Reachability in generalized product graphs is maintainable in* DynFO *under modifications to factors and transitions rules*[4] *for*
1. *a constant number of directed graphs under insertions and a constant number of transition rules,*
2. *polynomially many acyclic graphs and a constant number of transition rules,*
3. *polynomially many undirected graphs and polynomially many transition rules.*

Proof sketch. For (1) and (2), the key observation is that reachability in generalized graph products can be reduced to finding a solution of small natural numbers to a linear equation system. For proving (3), a characterization of reachability in generalized products of undirected graphs from [19] as well as the dynamic program for maitaining the rank of a matrix from [6] is used. Details can be found in the full paper. ◀

Observe that deciding reachability in generalized products of (1) polynomially many graphs with constant many transition rules and of (2) polynomially many acyclic graphs

[4] We permit single bit modifications to A, that is, modifying one bit of a transition rule at a time.

with polynomially many transitions rules are NP-hard problems. More precisely, the first generalizes reachability in the product of polynomially many graphs, which we already discussed above. As for the second, notice that the problem of deciding the existence of a 0–1 solution of a linear equation $A\vec{x} = \vec{1}$, which is known to be NP-hard even for a 0-1 matrix A [5, Chapter 8], can be straightforwardly reduced to reachability in the generalized product of acyclic graphs when polynomially many transition rules are allowed (by using the distance and linear equations characterization used in the proof of Theorem 19). These problems are thus unlikely to be maintainable in DynFO.

6 Conclusion

In this article we explored graph query languages in the dynamic descriptive complexity framework introduced independently by Dong, Su and Topor, and Patnaik and Immerman. Furthermore we investigated the strongly related question, under which conditions distances in graphs as well as reachability in product graphs can be maintained. Our work is only a first step towards a systematic understanding of graph queries in dynamic graph databases. In the following we discuss some interesting directions for further research.

For several restricted classes of graphs we exhibited first-order update programs for maintaining distances. We also showed that quantifier-free update formulas do not suffice. It remains open, whether distances can be maintained for general graphs; we conjecture that this is the case.

▶ **Open problem 1.** Exhibit a DynFO-program for maintaining distances.

As we have seen, reachability in products of labeled graphs is related to maintaining fragments of the graph query language ECRPQ. While we showed that reachability can be maintained in labeled products of acyclic graphs, this problem is already much harder for products of undirected, labeled paths – not to mention arbitrary labeled graphs.

▶ **Open problem 2.** Find dynamic DynFO-programs for maintaining reachability in products of restricted classes of labeled graphs.

Another interesting direction is to exhibit dynamic programs for other, more expressive query languages.

▶ **Open problem 3.** Identify further expressive query languages that can be maintained dynamically.

A candidate query language to be studied are nested regular expressions (NREs) [17]. NREs allow to express queries with some branching capabilities. For example, the NRE $(a[b])^*$ selects pairs of nodes that are connected by an a^*-labeled path such that every node on this path has an outgoing edge with label b. This query can easily be maintained in DynFO, as it is bounded first-order reducible to reachability. On the other hand, it is already unclear whether the query $(a[bc])^*$ can be maintained in DynFO.

Acknowledgements. We thank Pablo Barceló, Samir Datta and Thomas Schwentick for stimulating and illuminating discussions.

References

1 Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1, 2008.

2 Pablo Barceló Baeza. Querying graph databases. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA – June 22-27, 2013*, pages 175–188. ACM, 2013. URL: http://dl.acm.org/citation.cfm?id=2463664, doi:10.1145/2463664.2465216.

3 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

4 Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4):31, 2012. doi:10.1145/2389241.2389250.

5 Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., 2006.

6 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2015. doi:10.1007/978-3-662-47666-6_13.

7 Camil Demetrescu and Giuseppe F. Italiano. Mantaining dynamic matrices for fully dynamic transitive closure. *Algorithmica*, 51(4):387–427, 2008. doi:10.1007/s00453-007-9051-4.

8 Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In Catriel Beeri, Atsushi Ohori, and Dennis Shasha, editors, *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages – Object Models and Languages, Manhattan, New York City, USA, 30 August – 1 September 1993*, Workshops in Computing, pages 295–308. Springer, 1993.

9 Guozhu Dong and Rodney W. Topor. Incremental evaluation of datalog queries. In Joachim Biskup and Richard Hull, editors, *Database Theory – ICDT'92, 4th International Conference, Berlin, Germany, October 14-16, 1992, Proceedings*, volume 646 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 1992. doi:10.1007/3-540-56039-4_48.

10 Zvi Galil. Hierarchies of complete problems. *Acta Informatica*, 6(1):77–88, 1976.

11 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012. doi:10.1145/2287718.2287719.

12 Erich Grädel and Sebastian Siebertz. Dynamic definability. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT'12, Berlin, Germany, March 26-29, 2012*, pages 236–248. ACM, 2012. URL: http://dl.acm.org/citation.cfm?id=2274576, doi:10.1145/2274576.2274601.

13 William Hesse. The dynamic complexity of transitive closure is in DynTC0. *Theoretical Computer Science*, 296(3):473–485, 2003.

14 Peter Bro Miltersen. Cell probe complexity-a survey. In *19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 1999.

15 Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. Dynamic graph queries. *CoRR*, abs/1512.05511, 2015. URL: http://arxiv.org/abs/1512.05511.

16 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

17 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010. doi:10.1016/j.websem.2010.01.002.

18 Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008. doi:10.1137/060650271.

19 Dragan Stevanović. When is neps of graphs connected? *Linear Algebra and its Applications*, 301(1):137–144, 1999.

20 Volker Weber and Thomas Schwentick. Dynamic complexity theory revisited. *Theory Comput. Syst.*, 40(4):355–377, 2007. doi:10.1007/s00224-006-1312-0.

21 Peter T Wood. Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60, 2012.

22 Thomas Zeume. The dynamic descriptive complexity of k-clique. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 547–558. Springer, 2014. doi:10.1007/978-3-662-44522-8_46.

23 Thomas Zeume. *Small Dynamic Complexity Classes*. PhD thesis, TU Dortmund University, 2015.

24 Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, pages 38–49. OpenProceedings.org, 2014. URL: http://openproceedings.org/edbticdt2014/ICDT_toc.html, doi:10.5441/002/icdt.2014.08.

25 Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Inf. Comput.*, 240:108–129, 2015. doi:10.1016/j.ic.2014.09.011.

Verification of Evolving Graph-structured Data under Expressive Path Constraints

Diego Calvanese[1], Magdalena Ortiz[2], and Mantas Šimkus[2]

1 **Free University of Bozen-Bolzano, Bozen, Italy**
2 **TU Wien, Viena, Austria**

Abstract

Integrity constraints play a central role in databases and, among other applications, are fundamental for preserving data integrity when databases evolve as a result of operations manipulating the data. In this context, an important task is that of static verification, which consists in deciding whether a given set of constraints is preserved after the execution of a given sequence of operations, for every possible database satisfying the initial constraints. In this paper, we consider constraints over graph-structured data formulated in an expressive Description Logic (DL) that allows for regular expressions over binary relations and their inverses, generalizing many of the well-known path constraint languages proposed for semi-structured data in the last two decades. In this setting, we study the problem of static verification, for operations expressed in a simple yet flexible language built from additions and deletions of complex DL expressions. We establish undecidability of the general setting, and identify suitable restricted fragments for which we obtain tight complexity results, building on techniques developed in our previous work for simpler DLs. As a by-product, we obtain new (un)decidability results for the implication problem of path constraints, and improve previous upper bounds on the complexity of the problem.

1998 ACM Subject Classification H.2 [Database Management] General

Keywords and phrases Path constraints, Description Logics, Graph databases, Static verification

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.15

1 Introduction

Integrity constraints play a central role in databases and, among many other applications, are fundamental for preserving data integrity when databases evolve as a result of operations manipulating the data [1, 23, 6]. A fundamental problem in this context is *static verification*: given a set of integrity constraints, and a sequence of operations that describe changes on databases (over the same schema), the goal is to verify whether the constraints are *preserved* by the operations, that is, they are satisfied after their application, for *every* database that initially satisfies the constraints. This allows one to establish the acceptability of sequences of operations, which guarantees that applications maintain data integrity at runtime, independently of the specific database states that may be reached. However, static verification is very hard, and identifying sufficiently expressive languages for integrity constraints and data operations that allow for decidable verification is challenging.

In this paper, we consider *graph-structured data* (GSD), that is, relational data that contains unary and binary relations only, and thus admits a natural representation as a labeled graph. This data model is well suited for those settings where the data does not comply to a fixed schema, and the *topology* of the data relations is central. The GSD model became important already two decades ago due to the close relationship with semi-structured data [2, 13]. In the last decade it has gained renewed interest due to its relevance in the

Semantic Web and a diverse range of areas, including social networks, life-sciences, and program analysis, see e.g., [5, 36] and references therein. The study of query languages for GSD has been the focus of extensive research efforts in the database community over the last decade, based on the common consensus that GSD requires *navigational* query languages that allow, as minimal required functionality, to extract nodes that are connected by paths complying with a given regular language [36, 30]. The exploration of constraints for this data model is somehow more limited, partly due to the fact that even simple formalisms for constraining relations between regular paths result in undecidability of basic inference problems. Indeed, early proposals for path constraint languages are still viewed as adequate [3, 14, 16, 26, 17], but their wider adoption is hindered by the fact their implication problem is only known to be decidable under very strong restrictions. Recently there has been much interest in the study of containment for expressive query languages for GSD [27, 20]. In this basic form of static analysis, queries can be seen as expressing constraints over GSD, but also here inference turns out to be undecidable unless severe restrictions are imposed. For example, containment of *Graph-XPath* queries, a variation of XPath advocated for querying GSD, is undecidable in general, and has been shown decidable only when restricted to the so-called *path-positive* fragment [27]. In this paper we show undecidability for a path constraint language that is significantly more restricted than Graph-XPath, and even than the path constraints in [14], and improve the previous undecidability results that required to express paths that return to the initial point [14, 27].

We advocate an expressive Description Logic (DL) as constraint language for GSD. DLs are a family of languages tailored for representing structured knowledge, and for supporting inference over it [8]. They formalize domain knowledge by describing complex classes of objects, called *concepts*, and binary relations between them, called *roles*. Most DLs can be seen as (syntactic variants of) decidable fragments of classical first-order (FO) logic, or its extension with transitive closure. Different DLs provide different expressive means to describe knowledge, with the computational complexity of inference varying accordingly. DLs are the basis of state-of-the-art *ontology languages* for sharing domain conceptualizations [9], and they are naturally suited for describing data sources. They have been applied for the static analysis of traditional data models, such as UML class diagrams [10] and Entity Relationship schemata [7]. In the paradigm of *ontology based data access* [34, 28], which has gained great importance in the last decade, DL ontologies are used to describe possibly heterogeneous data sources, facilitating their management, and leveraging domain knowledge to improve access to them. Query answering and containment in this setting has been extensively studied, for a range of DLs and query languages [11, 15, 33].

In this paper we show that DLs are adequate also as constraint languages for GSD. We focus on the expressive DL $\mathcal{ZOI}$, also known as $\mathcal{ALCOI}b^{\mathsf{Self}}_{reg}$ [18], which features regular expressions over binary relations and their inverses, and allows for using them to impose complex relations between concepts and roles. $\mathcal{ZOI}$ can express the full path-positive fragment of Graph-XPath, and supports additional features that allow it to express even richer constraints on GSD. We also show that well-known path constraint languages proposed for GSD in the past [3, 14] can be naturally expressed in (variations of) $\mathcal{ZOI}$. Moreover, we can leverage results from the DL community to generalize and improve previous upper bounds on the complexity of the implication problem for decidable fragments of path constraints. This requires, however, to lift existing algorithms and complexity results for reasoning in $\mathcal{ZOI}$ to *finite* structures, since in the setting of verification of evolving GSD we are interested in *finite data instances*, and so far this DL had been studied over unrestricted, possibly infinite, models only [18]. This transfer of results to the finite setting is fortunately possible

since $\mathcal{ZOI}$ enjoys *finite model property* (FMP), as we are able to prove. This is a crucial stepping stone for our results, and an interesting contribution on its own right. Indeed, while the FMP has been long known for the closely related *converse PDL* [24, 21], $\mathcal{ZOI}$ has several further features. In particular, it allows for Boolean combinations of roles that make standard filtration techniques not directly applicable, and call for more subtle arguments that borrow ideas from FMP proofs for the two-variable fragment of first-order logic [32, 25].

For expressing operations on GSD we use the action language proposed in [4], that allows for the (possibly conditional) composition of basic operations that add or delete from a predicate the objects selected by a complex DL concept or role.[1] The undecidability of general path constraint implication implies that static verification becomes undecidable when if complex roles are allowed in actions. However, we regain decidability by restricting the roles in the action to be *simple roles* that allow for union, intersection, and difference of possibly inverse roles, but disallow composition and the Kleene star. Under these restrictions, we can rely on the techniques of [4] to reduce static verification in the presence of $\mathcal{ZOI}$ constraints to satisfiability of $\mathcal{ZOI}$ KBs, obtaining a tight ExpTime upper bound for the former.

The paper is organized as follows. In Section 2 we introduce the DL $\mathcal{ZOI}$, and establish the finite model property for it. Section 3 is devoted to path constraint languages. We tighten the undecidability of path constraint implication shown in [14]. We also generalize the decidability in [3] to a richer class, and improve the upper bound from 2ExpSpace to ExpTime by reducing the problem to reasoning in $\mathcal{ZOI}$. Section 4 studies the static verification of $\mathcal{ZOI}$ constraints over GSD, for actions expressed in the language proposed in [4]. We show that the problem is undecidable if arbitrary $\mathcal{ZOI}$ roles occur in actions, and impose suitable restrictions to obtain ExpTime decidability using the techniques of [4].

2 Expressive DLs for Expressing Constraints over GSD

In this paper, we formalize GSD as relational structures, which we call *instances*, over a unary and binary relational signature. We propose to use the rich DL $\mathcal{ZOI}$, also known as $\mathcal{ALCOI}b^{\mathsf{Self}}_{reg}$ [18], to express constraints over graph structured data. This is natural as, like other DLs, $\mathcal{ZOI}$ is defined over a relational vocabulary that contains unary and binary predicates only (respectively called *concept names* and *role names* in DL jargon) and the structures over which is interpreted are precisely GSD instances. A distinguishing feature of $\mathcal{ZOI}$, which makes it especially adequate for describing GSD, is that it can express relations between objects by allowing for *complex roles* defined using regular expressions.

▶ **Definition 1** ($\mathcal{ZOI}$ syntax)**.** We consider fixed, countably infinite sets $\mathsf{N_C}$ of *concept names*, $\mathsf{N_R}$ of *role names*, and $\mathsf{N_I}$ of individual names. We assume that the set $\mathsf{N_C}$ contains the special concepts $\top$ (top) and $\bot$ (bottom), while $\mathsf{N_R}$ contains the top (universal) role T and the bottom (empty) role B. We define ($\mathcal{ZOI}$) *atomic concepts* B, *concepts* C, C', *atomic roles* P, *simple roles* S, S', and *roles* R, R', where $a, b \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, and $r \neq \mathsf{T}$, according to the following syntax:

$$\begin{array}{rcl} B & \longrightarrow & A \mid \{a\} \\ C, C' & \longrightarrow & B \mid \neg C \mid C \sqcap C' \mid C \sqcup C' \mid \\ & & \forall R.C \mid \exists R.C \mid \exists S.\mathsf{Self} \end{array} \qquad \begin{array}{rcl} P & \longrightarrow & r \mid r^- \mid \{(a,b)\} \\ S, S' & \longrightarrow & P \mid S \cap S' \mid S \cup S' \mid S \setminus S' \\ R, R' & \longrightarrow & \mathsf{T} \mid \varepsilon \mid id(C) \mid S \mid R \cup R' \mid \\ & & R \circ R' \mid R^* \end{array}$$

We call $\mathcal{ZOI}$ expressions $\{a\}$ *nominal concepts*, and $\{(a,b)\}$ *nominal roles*. ◀

[1] In our setting, updates are performed data that is viewed as complete, and DL constraints are not used to infer new knowledge. Thus we do not run in the expressiveness issues considered e.g., in [31, 19].

We use $\mathcal{ZOI}$ concepts and roles to define a general form of knowledge bases, in which we allow for Boolean combinations of intensional and extensional level statements.

▶ **Definition 2** ($\mathcal{ZOI}$ knowledge bases)**.** A *concept inclusion* is an expression of the form $C \sqsubseteq C'$, where C, C' are arbitrary concepts, and a *role inclusion* is an expression of the form $S \sqsubseteq S'$, where S, S' are simple roles. An *assertion* is an expression of the form $C(a)$, $S(a, a')$, or $a \neq a'$, where C is a concept, S a simple role, and $\{a, a'\} \subseteq \mathsf{N_I}$. Then, *($\mathcal{ZOI}$) knowledge bases* (KBs) are defined inductively as follows:

(i) every inclusion and every assertion is a KB;
(ii) if $\mathcal{K}$, $\mathcal{K}'$ are KBs, so are $\mathcal{K} \wedge \mathcal{K}'$, $\mathcal{K} \vee \mathcal{K}'$, and $\dot{\neg}\mathcal{K}$. ◀

The semantics of $\mathcal{ZOI}$ is based on standard relational structures. An *instance* (or *interpretation*) $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty *domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that maps each individual $a \in \mathsf{N_I}$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, each concept name $A \in \mathsf{N_C}$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and each role name $r \in \mathsf{N_R}$ to a set $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, in such a way that $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, $\bot^{\mathcal{I}} = \emptyset$, $\mathsf{T}^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and $\mathsf{B}^{\mathcal{I}} = \emptyset$. The function $\cdot^{\mathcal{I}}$ is inductively extended to all $\mathcal{ZOI}$ concepts and roles as follows:

$$
\begin{array}{rl@{\qquad}rl}
\{a\}^{\mathcal{I}} &= \{a^{\mathcal{I}}\} & \{(a,b)\}^{\mathcal{I}} &= \{(a^{\mathcal{I}}, b^{\mathcal{I}})\} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & (r^-)^{\mathcal{I}} &= \{(y,x) \mid (x,y) \in r^{\mathcal{I}}\} \\
(C \sqcap C')^{\mathcal{I}} &= C^{\mathcal{I}} \cap C'^{\mathcal{I}} & (S \setminus S')^{\mathcal{I}} &= S^{\mathcal{I}} \setminus S'^{\mathcal{I}} \\
(C \sqcup C')^{\mathcal{I}} &= C^{\mathcal{I}} \cup C'^{\mathcal{I}} & (S \cap S')^{\mathcal{I}} &= S^{\mathcal{I}} \cap S'^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\} & (R \cup R')^{\mathcal{I}} &= R^{\mathcal{I}} \cup R'^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} & (R \circ R')^{\mathcal{I}} &= R^{\mathcal{I}} \circ R'^{\mathcal{I}} \\
(\exists S.\mathrm{Self})^{\mathcal{I}} &= \{x \mid (x,x) \in S^{\mathcal{I}}\} & (R^*)^{\mathcal{I}} &= (R^{\mathcal{I}})^* \\
 & & (id(C))^{\mathcal{I}} &= \{(x,x) \mid x \in C^{\mathcal{I}}\} \\
 & & (\varepsilon)^{\mathcal{I}} &= \{(x,x) \mid x \in \Delta^{\mathcal{I}}\}
\end{array}
$$

where $\cap$, $\cup$, and $\setminus$ are overloaded to denote also the standard set-theoretic operations, $\circ$ to denote composition, and $\cdot^*$ to denote the reflexive transitive closure of a binary relation.

$\mathcal{I}$ *satisfies* the inclusion $E \sqsubseteq E'$ if $E^{\mathcal{I}} \subseteq E'^{\mathcal{I}}$, the assertions $A(a)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$, $S(a,b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in S^{\mathcal{I}}$, and $a \neq b$ if $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Satisfaction is extended in the usual way to KBs, which are Boolean combinations of inclusions and assertions. When $\mathcal{I}$ satisfies $\mathcal{K}$, we also say that $\mathcal{I}$ is a *model* of $\mathcal{K}$, and denote it with $\mathcal{I} \models \mathcal{K}$.

As basic reasoning task we consider *KB satisfiability*, which consists in deciding, given a KB $\mathcal{K}$, whether $\mathcal{K}$ admits a model. Other standard reasoning tasks, like concept (resp., role) satisfiability, that is, deciding whether there exists an interpretation where the extension of a given concept (resp., role) is not empty, can be reduced to KB satisfiability.

▶ Remark. The roles ε and $\{(a,b)\}$, which are not usually included in $\mathcal{ZOI}$, are just syntactic sugar. Indeed, ε has the same meaning as $id(\top)$ and captures the identity relation. Nominal roles $\{(a,b)\}$ can be easily simulated in $\mathcal{K}$ by replacing each occurrence of $\{(a,b)\}$ by a fresh role name r_{ab}, and conjunctively adding to $\mathcal{K}$ the KB $r_{ab}(a,b) \wedge (\exists r_{ab}.\top \sqsubseteq \{a\}) \wedge (\top \sqsubseteq \forall r_{a,b}.\{b\})$. This ensures that $r_{ab}^{\mathcal{I}} = \{(a^{\mathcal{I}}, b^{\mathcal{I}})\}$ in every model of the modified KB.

▶ **Example 3.** As a running example, we consider the following self-explanatory instance $\mathcal{I}_{\mathsf{Uni}}$. For simplicity, in the examples we interpret individuals as themselves (i.e., we make the standard name assumption).

$$
\begin{aligned}
\mathsf{Dept}^{\mathcal{I}} &= \{\text{CS\_Dept}\} \\
\mathsf{Program}^{\mathcal{I}} &= \{\text{BSc\_CSci}, \text{MSc\_CompLogic}, \text{MSc\_Bioinformatics}\} \\
\mathsf{Course}^{\mathcal{I}} &= \{\text{DataStruct:CS202}, \text{FoundDBs:CS327}, \text{DLs:CS451}\} \\
\mathsf{partOf}^{\mathcal{I}} &= \{(\text{DLs:CS451}, \text{mod\_KR})\} \\
\mathsf{offers}^{\mathcal{I}} &= \{(\text{CS\_Dept}, \text{BSc\_CompSci}), (\text{CS\_Dept}, \text{MSc\_CompLogic}), \\
&\quad (\text{CS\_Dept}, \text{MSc\_Bioinformatics})\{(\text{CS\_Dept}, \text{DataStruct:CS202}), \\
&\quad (\text{CS\_Dept}, \text{FoundDBs:CS327}), (\text{CS\_Dept}, \text{DLs:CS451})\}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{requires}^{\mathcal{I}} = \{&(\text{BSc\_CSci}, \text{DataStruct:CS202}), (\text{MSc\_CompLogic}, \text{FoundDBs:CS327}), \\
&(\text{MSc\_CompLogic}, \text{mod\_KR}), (\text{MSc\_Bioinformatics}, \text{DLs:CS451})\}
\end{aligned}
$$

Consider the KB $\mathcal{K}_{\mathsf{Uni}}$ defined as the conjunction of the following $\mathcal{ZOI}$ constraints: ϕ_1 says that the domain of 'offers' are the departments, and ϕ_2 says that its range is the union of programs and courses. Similarly, ϕ_3 and ϕ_4 restrict the domain of 'requires' to programs, and its range to courses other entities that comprise courses, like modules. Finally ϕ_5 says that every course that is required (directly, or because it is part of a required module) must be offered.

$$
\begin{aligned}
\phi_1 &= \exists \mathsf{offers}.\top \sqsubseteq \mathsf{Dept} & \phi_2 &= \top \sqsubseteq \forall \mathsf{offers}.(\mathsf{Program} \sqcup \mathsf{Course}) \\
\phi_3 &= \exists \mathsf{requires}.\top \sqsubseteq \mathsf{Program} & \phi_4 &= \top \sqsubseteq \forall \mathsf{requires}.(\exists \mathsf{partOf}^{-*}.\mathsf{Course}) \\
\phi_5 &= \mathsf{Course} \sqcap \exists(\mathsf{partOf}^{*} \circ \mathsf{requires}^{-}).\top \sqsubseteq \exists \mathsf{offers}^{-}.\top
\end{aligned}
$$

Note that all these constraints are satisfied by our instance, that is, $\mathcal{I}_{\mathsf{Uni}} \models \mathcal{K}_{\mathsf{Uni}}$.

We note that $\mathcal{ZOI}$ is closely related to *path-positive Graph-XPath* (abbreviated $\mathsf{GXPath}_{reg}^{path\text{-}pos}$) introduced in [30]. By viewing arc labels as role names, node formulas in $\mathsf{GXPath}_{reg}^{path\text{-}pos}$ can be written as $\mathcal{ZOI}$ concepts, and $\mathsf{GXPath}_{reg}^{path\text{-}pos}$ path formulas as $\mathcal{ZOI}$ roles. Additionally $\mathcal{ZOI}$ extends $\mathsf{GXPath}_{reg}^{path\text{-}pos}$ with other features, such as Boolean combinations of node and path labels (i.e., Boolean concepts and roles), nominals, and concepts of the form $\exists S.\text{Self}$.

In [18], a tree-automata based algorithm for checking satisfiability of $\mathcal{ZOI}$ concepts is provided, and by using a variant of *internalization* [35], this is exploited to check satisfiability of $\mathcal{ZOI}$ KBs constituted by a conjunction of (positive) assertions and inclusions. It is easy to extend internalization also to $\mathcal{ZOI}$ KBs of the more general form considered here, and thus reduce satisfiability of a $\mathcal{ZOI}$ KB to satisfiability of a $\mathcal{ZOI}$ concept. The proof is given in the extended version of the paper.

▶ **Theorem 4.** *Given a $\mathcal{ZOI}$ KB $\mathcal{K}$, one can construct in linear time a $\mathcal{ZOI}$ concept $C_{\mathcal{K}}$ such that $\mathcal{K}$ is satisfiable if and only if $C_{\mathcal{K}}$ is so.*

From this result and the ExpTime upper bound for concept satisfiability given in [18], it follows immediately that satisfiability of $\mathcal{ZOI}$ KBs is decidable in single exponential time. This is worst-case optimal, since the problem is ExpTime-hard even for significantly simpler DLs like $\mathcal{ALC}$ [8].

▶ **Theorem 5** ([18])**.** *Checking satisfiability of $\mathcal{ZOI}$ KBs is an* ExpTime*-complete problem.*

Finite Model Reasoning in $\mathcal{ZOI}$. In the setting of GSD we are usually interested in finite instances. In the DL literature, however, finite model reasoning has received significantly less attention than reasoning with respect to unrestricted models. To our knowledge, finite model reasoning for $\mathcal{ZOI}$ has not been addressed so far. However, as we show in the following, $\mathcal{ZOI}$ enjoys the *finite model property*, which states that every satisfiable KB admits a model whose domain is finite. In line with what has been done for other logics that cannot express functionality, keys, or number restrictions, we can show this through a filtration argument [24, 21]. However, due to the presence of both transitive closure over roles and role intersection and difference, the proof is more involved than for logics that involve none or only one of the two kinds of constructs.

We say a KB $\mathcal{K}$ is *finitely satisfiable* if it admits a finite model, i.e., a model with a finite domain. The proof of the following result is given in the extended version of the paper.

▶ **Theorem 6.** *Let $\mathcal{K}$ be a $\mathcal{ZOI}$ KB. Then $\mathcal{K}$ is satisfiable if and only if $\mathcal{K}$ is finitely satisfiable.*

3 Path Constraints

We define a language for path constraints inspired by [3, 14, 26] and closely related to $\mathcal{ZOI}$.

▶ **Definition 7** (Path constraints)**.** A *path constraint* φ has the form $[R_p](R_\ell \subseteq R_r)$, where R_p, R_ℓ, and R_r are arbitrary $\mathcal{ZOI}$ roles. The role R_p is called *prefix* of φ, while the roles R_ℓ and R_r are respectively called the *left tail* and the *right tail* of φ. If $R_p = \varepsilon$, we call φ a *prefix-empty* constraint[2], and write it simply as $R_\ell \subseteq R_r$. ◀

This definition generalizes the well-known path constraint languages from [3, 14]. A complex role R built from the symbols in $\mathsf{N_R} \cup \{\varepsilon\}$ using $\circ$, $\cup$, and * is called a *(one-way) regular path role*, and if additionally it does not contain $\cup$ or * then it is called a *(one-way) word role*. A *one-way regular path constraint* (called simply *path constraint* in [3]) is a prefix-empty constraint where R_ℓ and R_r are one-way regular path roles. If, in addition, R_ℓ and R_r are one-way word roles, the path constraint is called a *word constraint* in [3]. The language of path constraints in [14] allows for non-empty prefixes, but restricts the left tail to be a one-way word role, and the right tail to be either a one-way word role (in the so-called *forward constraints*), or an *inverted one-way word role* (in *backward constraints*), which is a sequence of concatenated inverses of role names (that is, $r_1^- \circ \cdots \circ r_n^-$ with $n \geq 0$).[3]

Now we define the semantics of path constraints and their fundamental reasoning problem, namely *implication* of path constraints, both in its finite and in its unrestricted variants.

In the semantics of early path constraint languages [3, 14], every instance has a distinguished root object at which the constraints are enforced. We introduce a minor variation of this semantics, which we call *pointed* semantics, where rather than a fixed name for the root node, we allow for any individual name to be used as its identifier. Later works advocated what we call the *global* semantics [22, 26], in which constraints are enforced at every point in the model, rather than at just one. We note that the global semantics is in general computationally more costly, and causes undecidability of the implication problem for some fragments that are decidable under the pointed semantics [3, 14]. We discuss below how both

[2] *Prefix-empty* constraints were called *simple* in [14]. We use a different name to avoid confusion with the *simple roles* of Definition 1.

[3] We note that [14] uses a different syntax with explicit variables.

semantics can be naturally captured in DLs, and provide decidability and undecidability results for both of them.

▶ **Definition 8** (Pointed and rooted semantics, implication problem). Let $\varphi = [R_p](R_\ell \subseteq R_r)$ be a path constraint. For an interpretation $\mathcal{I}$, we let $\varphi^{\mathcal{I}}$ be the set of objects $d \in \Delta^{\mathcal{I}}$ such that for each $d', d'' \in \Delta^{\mathcal{I}}$, if $(d, d') \in R_p^{\mathcal{I}}$ and $(d', d'') \in R_\ell^{\mathcal{I}}$, then $(d', d'') \in R_r^{\mathcal{I}}$.

A *pointed instance* is a pair $\mathcal{I}, a$ of an instance $\mathcal{I}$ and an individual a. We call $\mathcal{I}, a$ a *pointed model* of φ, and write $\mathcal{I}, a \models \varphi$ if $a^{\mathcal{I}} \in \varphi^{\mathcal{I}}$. Similarly, we write $\mathcal{I}, a \models \Gamma$ for a set Γ of constraints, if $\mathcal{I}, a \models \varphi$ for each $\varphi \in \Gamma$. We write $\Gamma, a \models \varphi$ if $\mathcal{I}, a \models \varphi$ for every pointed model $\mathcal{I}, a$ of Γ, and write $\Gamma, a \models_{\mathit{fin}} \varphi$ if $\mathcal{I}, a \models \varphi$ for every *finite* pointed model $\mathcal{I}, a$ of Γ. The *(finite) pointed implication problem* consists in deciding, given an individual a, a set Γ of path constraints, and a path constraint φ, whether $\Gamma, a \models_{(\mathit{fin})} \varphi$.

Let $\varphi = [R_p](R_\ell \subseteq R_r)$ be a path constraint. We call $\mathcal{I}$ a *global model* of φ, and write $\mathcal{I} \models \varphi$ if $\varphi^{\mathcal{I}} = \Delta^{\mathcal{I}}$. We write $\mathcal{I} \models \Gamma$ for a set Γ of constraints, if $\mathcal{I} \models \varphi$ for each $\varphi \in \Gamma$. We write $\Gamma \models \varphi$ if $\mathcal{I} \models \varphi$ for every $\mathcal{I}$ with $\mathcal{I} \models \Gamma$, and write $\Gamma \models_{\mathit{fin}} \varphi$ if $\mathcal{I} \models \varphi$ for every *finite* $\mathcal{I}$ with $\mathcal{I} \models \Gamma$. The *(finite) global implication problem* consists in deciding, given a set Γ of path constraints and a path constraint φ, whether $\Gamma \models_{(\mathit{fin})} \varphi$. ◀

▶ **Example 9.** Consider the constraint $\varphi_1 = R_1 \subseteq R_2$, where
$R_1 = \mathit{id}(\mathsf{Dept}) \circ \mathsf{partOf}^{-*} \circ \mathsf{offers} \circ \mathsf{requires}^{-} \circ \mathsf{partOf}^{-*}$
$R_2 = \mathit{id}(\mathsf{Dept}) \circ \mathsf{partOf}^{-*} \circ \mathsf{offers}$
Intuitively, (a node interpreting) a department satisfies φ_1 if every course required by a program offered by (a suborganization of) the department is offered by (a suborganization of) the same department. With the rooted semantics, we can enforce the constraint for some specific departments. For example, we may require it for computer science, and our example instance satisfies it: $\mathcal{I}_{\mathsf{Uni}}, \mathrm{CS\_Dept} \models \varphi_1$. With the global semantics, the constraint would apply to all departments, but we can easily modify it so that it applies only to the desired departments, e.g., $(\mathit{id}(\mathrm{CS\_dept}) \cdot R_1) \subseteq R_2$.

To illustrate the use of prefixes, suppose that the policy expressed by φ_1 is enforced not at the department level, but at the higher faculty/school level. For example, suppose that the School of Science and Engineering requires that all departments offer within their department every mandatory course in their programs (while other schools may allow for mandatory courses that are offered by different departments). This is captured by the constraint with non-empty prefix $\varphi_1 = [\mathit{id}(\mathsf{School\_SciEng}) \circ \mathsf{hasDepartment}](R_1 \subseteq R_2)$.

Expressing Path Constraints in $\mathcal{ZOI}$ with Role Difference. To express path constraints, we extend $\mathcal{ZOI}$ with difference $R \setminus R'$ of arbitrary roles, resulting in the logic we call $\mathcal{ZOI}^{\setminus}$.

▶ **Definition 10.** $\mathcal{ZOI}^{\setminus}$ roles are defined analogously to $\mathcal{ZOI}$ roles, except that for complex roles we have $R, R' \longrightarrow \mathsf{T} \mid \mathit{id}(C) \mid S \mid R \cup R' \mid R \circ R' \mid R \setminus R' \mid R^*$.
The syntax and semantics of $\mathcal{ZOI}^{\setminus}$ concepts, assertions, axioms, and knowledge bases are defined as for $\mathcal{ZOI}$, but allowing for $\mathcal{ZOI}^{\setminus}$ roles in the place of $\mathcal{ZOI}$ roles. ◀

Entailment of path constraints defined above can be reduced to reasoning in $\mathcal{ZOI}^{\setminus}$:

▶ **Lemma 11.** *For a path constraint $\varphi = [R_p](R_\ell \subseteq R_r)$, let $C_\varphi = \forall R_p.(\forall(R_\ell \setminus R_r).\bot)$. Then, for every instance $\mathcal{I}$, we have that $\varphi^{\mathcal{I}} = C_\varphi^{\mathcal{I}}$. Consequently, for a set Γ of path constraints, a*

path constraint φ, and $a \in \mathsf{N_I}$, we have:

$$\begin{array}{lll}
\Gamma, a \models \varphi & \textit{iff} & \big(\prod_{\gamma\in\Gamma} C_\gamma \sqcap \neg C_\varphi\big)(a) \quad \textit{is unsatisfiable,} \\
\Gamma, a \models_{\mathit{fin}} \varphi & \textit{iff} & \big(\prod_{\gamma\in\Gamma} C_\gamma \sqcap \neg C_\varphi\big)(a) \quad \textit{is finitely unsatisfiable,} \\
\Gamma \models \varphi & \textit{iff} & \bigwedge_{\gamma\in\Gamma}(\top \sqsubseteq C_\gamma) \wedge \dot{\neg}(\top \sqsubseteq C_\varphi) \quad \textit{is unsatisfiable,} \\
\Gamma \models_{\mathit{fin}} \varphi & \textit{iff} & \bigwedge_{\gamma\in\Gamma}(\top \sqsubseteq C_\gamma) \wedge \dot{\neg}(\top \sqsubseteq C_\varphi) \quad \textit{is finitely unsatisfiable.}
\end{array}$$

We will see that, unfortunately, both implication of path constraints and reasoning in $\mathcal{ZOI}^{\setminus}$ are undecidable. Before moving to these negative results, though, we point out that the lemma above implies that the upper bounds for reasoning in plain $\mathcal{ZOI}$ extend to the implication of path constraints γ where only simple $\mathcal{ZOI}$ roles occur. Since in this case the resulting C_γ is a $\mathcal{ZOI}$ concept, from Lemma 11 and theorem 5 we get:

▶ **Corollary 12.** *Let Γ be a set of path constraints and φ a path constraint such that, for each $\gamma = [R_p](R_\ell \subseteq R_r) \in \Gamma \cup \{\varphi\}$, R_p, R_ℓ and R_r are all simple $\mathcal{ZOI}$ roles. Then $\Gamma, a \models \varphi$, $\Gamma, a \models_{\mathit{fin}} \varphi$, $\Gamma \models \varphi$, and $\Gamma \models_{\mathit{fin}} \varphi$ are all decidable in* ExpTime.

Undecidability of Path Constraint Implication. Unfortunately, both the finite and the unrestricted implication problems are undecidable in rather restricted settings. The following result was established already several years ago:[4]

▶ **Theorem 13** ([14])**.** *Assume that every instance has a distinguished* root *element o, and that there is some $a_o \in \mathsf{N_I}$ such that $a_o^{\mathcal{I}} = o$ in every $\mathcal{I}$. The problem of checking whether $\Gamma, a_o \models \varphi$ and whether $\Gamma, a_o \models_{\mathit{fin}} \varphi$ are undecidable, even when φ and all constraints in Γ satisfy one of the following two restrictions:*

- *All prefixes, left tails, and right tails are one-way word roles (that is, only forward constraints according to [14] are allowed).*
- *All prefixes and left tails are one-way word roles different from ε, and each right tail is a one-way word role or an inverted one-way word role, and is different from ε.*

We strengthen this result, showing undecidability when both restrictions apply: only one-way word roles of length one or two are allowed. Our proof encodes a Turing rather than a two-register machine as in [14], and we believe some readers may find it simpler.

▶ **Theorem 14.** *The problems of checking whether $\Gamma, a \models \varphi$ and whether $\Gamma, a \models_{\mathit{fin}} \varphi$, given Γ, a, and φ are undecidable. This holds even when φ is of the form $r_1 \subseteq r_2$ and Γ contains only constraints of the following forms, where $\{r, r_1, r_2, r_3\} \subseteq \mathsf{N_R}$:*

$$r_1 \circ r_2 \subseteq r_3 \qquad r_1 \subseteq r_2 \circ r_3 \qquad [r](r_1 \circ r_2 \subseteq r_3) \qquad [r](r_1 \subseteq r_2 \circ r_3)$$

Proof. As in [14] we employ the notion of *conservative reduction classes* to simultaneously deal with general and finite implication. In addition, we see deciding $\Gamma, a \models \varphi_{\mathcal{M}}$ as checking unsatisfiability of the first order formula that corresponds to Γ and the negation of $\varphi_{\mathcal{M}}$. The same is true for finite implication. Let FO denote the set of FO formulae, X be a recursive subset of FO, and let $f : FO \to X$ be a recursive function such that:

- if $\beta \in FO$ is unsatisfiable, then $f(\beta)$ is unsatisfiable, and
- if $\beta \in FO$ has a finite model, then $f(\beta)$ has a finite model.

[4] In fact, the authors of [14] show that implication is r.e. complete, and finite implication co-r.e. complete.

Then X is a conservative reduction class and thus satisfiability of formulae in X is co-r.e.-complete and finite satisfiability r.e.-complete [12]. It is well known that for a first-order formula β we can build a procedure that takes no input and terminates iff β is unsatisfiable or has a finite model. Thus to show the undecidability of general and finite implication, it suffices to show how the computation of such a procedure can be simulated using path constraints. We consider a deterministic Turing Machine (TM) $\mathcal{M}$ with state set Q and alphabet Σ. We assume that $\mathcal{M}$ has two designated states q^1_{fin} and q^2_{fin}. We build Γ and $\varphi_{\mathcal{M}}$ such that the following conditions are satisfied:

1. If $\mathcal{M}$ reaches q^1_{fin} starting from the empty string as input, then $\Gamma_{\mathcal{M}}, a \models \varphi_{\mathcal{M}}$.
2. If $\mathcal{M}$ reaches q^2_{fin} starting from the empty string as input, then there exists a finite $\mathcal{I}$ such that $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$ and $\mathcal{I}, a \not\models \varphi_{\mathcal{M}}$.

We assume that $\mathcal{M}$ starts at the initial tape position, and never moves to the left of it. Moreover, the tape is initially empty, that is, it only contains the blank symbol $\sqcup$. The transition function of $\mathcal{M}$ is of the form $\delta : Q \times \Sigma \to \Sigma \times Q \times \{\mathrm{R}, \mathrm{L}\}$, with the usual reading, where R and L stand for right and left, respectively. The initial state of M is q_{ini}.

The constraint $\varphi_{\mathcal{M}}$ takes the form $u_{\mathsf{ini}} \subseteq u_{\mathsf{halt}}$ where u_{ini} and u_{halt} are two role names. We define the set $\Gamma_{\mathcal{M}}$ next. Intuitively, the idea of the reduction is the following. Assume an arbitrary pointed structure $\mathcal{I}, a$ and let o denote $a^{\mathcal{I}}$. Whenever there is some $o' \in \Delta^{\mathcal{I}}$ such that $(o, o') \in u^{\mathcal{I}}_{\mathsf{ini}}$, the constraints in $\Gamma_{\mathcal{M}}$ will ensure that if $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$, then $\mathcal{I}$ contains a (possibly infinite) structure that represents the computation of $\mathcal{M}$, and that $(o, o') \in u^{\mathcal{I}}_{\mathsf{ini}}$ whenever the computation halts.

To provide an intuitive description of how this structure is enforced, we will use the term *r-arc* to refer to a pairs $(d, d') \in r^{\mathcal{I}}$ for a role name r. Now we describe the constraints in $\Gamma_{\mathcal{M}}$, which ensure that from the initial arc u_{ini} we build a full computation of $\mathcal{M}$. We use role names of the form $t_{q,\sigma}$ and $f_{q,\sigma}$ for each $q \in Q \cup \{\#\}$ and each $\sigma \in \Sigma \cup \{\sqcup\}$. Each of these symbols stands for a tape position containing the symbol σ. The marker $\#$ indicates that the head of $\mathcal{M}$ is not on the current position, while $q \in Q$ indicates that the head is on the current position and $\mathcal{M}$ is in state q. The symbols $f_{q,\sigma}$ are used to distinguish the right-most tape position, while regular 'inner' positions are represented by symbols $t_{q,\sigma}$. The first two constraints are prefix-empty and use the auxiliary role u_{aux}. They ensure that whenever there is an u_{ini} arc, there exists also an arc labeled $f_{q_{\mathsf{ini}},\sqcup}$, indicating that $\mathcal{M}$ is in state q_{ini}, the current tape position contains the symbol $\sqcup$, and the current tape position is the right-most one that has been visited so far:

$$u_{\mathsf{ini}} \subseteq u_{\mathsf{aux}} \circ u_{\mathsf{out}} \tag{1}$$

$$u_{\mathsf{aux}} \subseteq u_{\mathsf{in}} \circ f_{q_{\mathsf{ini}},\sqcup} \tag{2}$$

Note that there is an u_{in} arc from o to the beginning of $f_{q_{\mathsf{ini}},\sqcup}$, and an u_{out} arc from its end to o'. For the initial and final points of this $f_{q_{\mathsf{ini}},\sqcup}$ arc, and of all the arcs r that our construction will generate below in order to simulate the runs of $\mathcal{M}$, we want the role name u_{in} to connect o to the point, and u_{out} to connect the point to o'. We ensure this by adding the following constraints:

$$(u_{\mathsf{in}} \circ r \subseteq u_{\mathsf{in}}) \quad \text{for every } r \in \mathsf{N_R} \setminus \{u_{\mathsf{ini}}, u_{\mathsf{halt}}, u_{\mathsf{in}}, u_{\mathsf{out}}, u_{\mathsf{aux}},\} \text{ occurring in } \Gamma_{\mathcal{M}} \tag{3}$$

$$[u_{\mathsf{in}}](r \circ u_{\mathsf{out}} \subseteq u_{\mathsf{out}}) \quad \text{for every } r \in \mathsf{N_R} \setminus \{u_{\mathsf{ini}}, u_{\mathsf{halt}}, u_{\mathsf{in}}, u_{\mathsf{out}}, u_{\mathsf{aux}},\} \text{ occurring in } \Gamma_{\mathcal{M}} \tag{4}$$

Note that there is a pair of these constraints for every role name occurring in the rest of the proof, except for the u roles that do not have a direct correspondence with the configurations of $\mathcal{M}$. With these axioms, the initial and final points of the $f_{q_{\mathsf{ini}},\sqcup}$ arc are both reachable

Creating the arc $f_{q_{\mathsf{ini}},\sqcup}$ for the first tape position (constraints (1) and (2))

Connecting the new arc to the spy-points (constraints (3) and (4) for $r = f_{q_{\mathsf{ini}},\sqcup}$)

Figure 1 Model of $\Gamma_{\mathcal{M}}$ and $u_{\mathsf{ini}}(o, o')$ representing the initial configuration of $\mathcal{M}$.

from o, and both reach o'. The same holds for any other arc implied by the constraints below. The intended model of the constraints we have described so far is depicted in Figure 1.

Now we give the constraints that ensure that the computation of $\mathcal{M}$ is correctly simulated. The core idea is that (the initial and final points of the arc representing) each tape position will be linked via an n role (for $n$ext configuration) to (the initial and final points of an arc representing) the same tape position in the following configuration. Differently subindexed n roles and auxiliary '$d$iagonal' d roles are used to propagate information between configurations.

The first group handles the case where the machine is at the right-most visited tape position (that is, there is a symbol $f_{q\sigma}$), and executes a transition that moves to the right:

$$[u_{\mathsf{in}}](f_{q\sigma} \subseteq d_{\leftarrow \# \sigma'} \circ n_{fq'}) \qquad \text{for each } \delta(q,\sigma) = (q', \sigma', \mathrm{R}) \tag{5}$$

$$[u_{\mathsf{in}}](n_{fq} \subseteq f_{q\sqcup} \circ d_f) \qquad \text{for each } q \in Q \tag{6}$$

After the tape contents have been updated at the current position, and the automaton has changed to the new state and moved right to a new final tape position, it is only left to go leftwards propagating to the next configuration the remaining tape contents. This is ensured by the following axioms:

$$[u_{\mathsf{in}}](t_{q\sigma} \circ n_{\leftarrow} \subseteq d_{\leftarrow q\sigma}) \qquad \text{for each } q \in Q \cup \{\#\}, \sigma \in \Sigma \cup \{\sqcup\} \tag{7}$$

$$[u_{\mathsf{in}}](d_{\leftarrow q\sigma} \subseteq n_{\leftarrow} \circ t_{q\sigma}) \qquad \text{for each } q \in Q \cup \{\#\}, \sigma \in \Sigma \cup \{\sqcup\} \tag{8}$$

The next group handles also the case where the machine is currently at the right-most visited tape position indicated by $f_{q\sigma}$, but this time it executes a transition that moves to the left:

$$[u_{\mathsf{in}}](f_{q\sigma} \subseteq d_{\leftarrow q' f \# \sigma'} \circ n_f) \qquad \text{for each } \delta(q,\sigma) = (q', \sigma', \mathrm{L}) \tag{9}$$

$$[u_{\mathsf{in}}](d_{\leftarrow q f \# \sigma} \subseteq n_{\leftarrow q} \circ f_{\# \sigma}) \qquad \text{for each } q \in Q, \sigma \in \Sigma \cup \{\sqcup\} \tag{10}$$

$$[u_{\mathsf{in}}](t_{\# \sigma} \circ n_{\leftarrow q} \subseteq d_{\leftarrow q\sigma}) \qquad \text{for each } q \in Q, \sigma \in \Sigma \cup \{\sqcup\} \tag{11}$$

We recall that constraints (7) and (8) already ensure that the remaining tape contents are properly propagated. Next we handle transitions to the right, from a non-final tape position:

$$[u_{\mathsf{in}}](t_{q\sigma} \subseteq d_{\leftarrow \# \sigma'} \circ n_{q'\rightarrow}) \qquad \text{for each } \delta(q,\sigma) = (q', \sigma', \mathrm{R}) \tag{12}$$

$$[u_{\mathsf{in}}](n_{q\rightarrow} \circ t_{\# \sigma} \subseteq d_{q\sigma\rightarrow}) \qquad \text{for each } q \in Q, \sigma \in \Sigma \cup \{\sqcup\} \tag{13}$$

Additionally to the contents to the left, which have been taken care of, we also need to propagate tape contents to the right of the current position. We use constraints analogous to (7) and (8):

$$[u_{\mathsf{in}}](n_{\rightarrow} t_{q\sigma} \subseteq d_{q\sigma\rightarrow}) \qquad \text{for each } q \in Q \cup \{\#\}, \sigma \in \Sigma \cup \{\sqcup\} \tag{14}$$

$$[u_{\mathsf{in}}](d_{q\sigma\rightarrow} \subseteq t_{q\sigma} n_{\rightarrow}) \qquad \text{for each } q \in Q \cup \{\#\}, \sigma \in \Sigma \cup \{\sqcup\} \tag{15}$$

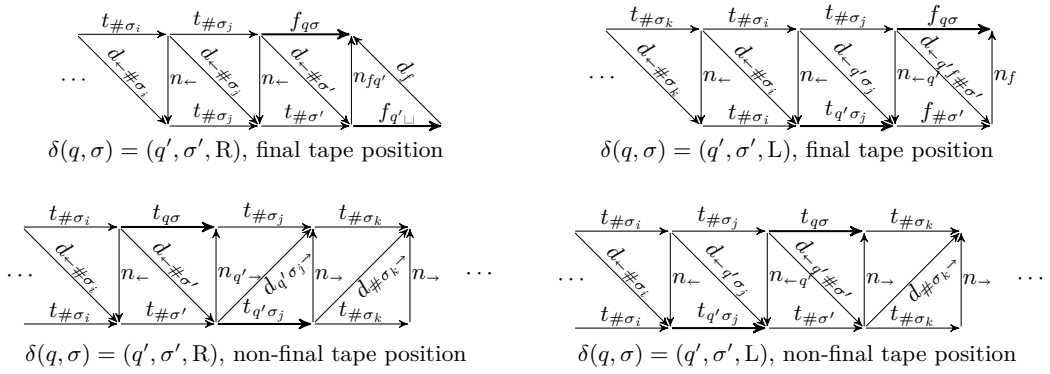


Figure 2 Parts of models of $\Gamma_{\mathcal{M}}$ and $u_{\mathsf{ini}}(o, o')$ representing the transitions of $\mathcal{M}$.

When we propagate to the right, we need to ensure that the final position $t_{q\sigma}$ is also copied to the next configuration:

$$[u_{\mathsf{in}}](n_{\rightarrow} \circ f_{q\sigma} \subseteq d_{q\sigma f}) \qquad \text{for each } q \in Q \cup \{\#\}, \sigma \in \Sigma \cup \{\sqcup\} \tag{16}$$

$$[u_{\mathsf{in}}](d_{q\sigma f} \subseteq f_{q\sigma} \circ n_f) \qquad \text{for each } q \in Q \cup \{\#\}, \sigma \in \Sigma \cup \{\sqcup\} \tag{17}$$

Finally we handle transitions to the left from a non-final tape position:

$$[u_{\mathsf{in}}](t_{q\sigma} \subseteq d_{\leftarrow q'\#\sigma'} \circ n_{\rightarrow}) \qquad \text{for each } \delta(q,\sigma) = (q', \sigma', \mathrm{L}) \tag{18}$$

$$[u_{\mathsf{in}}](d_{\leftarrow q\#\sigma} \subseteq n_{\leftarrow q} \circ t_{\#\sigma}) \qquad \text{for each } q \in Q, \sigma \in \Sigma \cup \{\sqcup\} \tag{19}$$

$$[u_{\mathsf{in}}](t_{\#\sigma} \circ n_{\leftarrow q} \subseteq d_{\leftarrow q\sigma}) \qquad \text{for each } q \in Q, \sigma \in \Sigma \cup \{\sqcup\} \tag{20}$$

Figure 2 illustrates how the constraints simulate the transitions of $\mathcal{M}$. For readability, the u_{in} arcs to and u_{out} arcs from every node are omitted.

An interpretation that is a model of the transitions we have presented correctly executes a computation of $\mathcal{M}$. It is only left to enforce the special role u_{halt} to hold between o and o' iff the computation halts in q^1_{fin}. We first ensure that if an arc $t_{q^1_{\mathsf{fin}},\sigma}$ or $f_{q^1_{\mathsf{fin}},\sigma}$ for some $\sigma \in \Sigma \cup \{\sqcup\}$ was generated (that is, q^1_{fin} was reached in the computation), an u_{halt} arc is created; note that if $\mathcal{M}$ halts in q^2_{fin} nothing enforces u_{halt} and there will be a model of the constraints where φ_M does not hold. Then we ensure that if there is an u_{halt} arc anywhere, then there is such an arc between o and o'. This is easy to do, exploiting the fact that all points are reachable via u_{ini} from o, and reach o' via u_{in}:

$$u_{\mathsf{in}} \circ t_{q^1_{\mathsf{fin}}\sigma} \subseteq u_{\mathsf{halt}} \qquad \text{for each } \sigma \in \Sigma \cup \{\sqcup\} \tag{21}$$

$$u_{\mathsf{in}} \circ f_{q^1_{\mathsf{fin}}\sigma} \subseteq u_{\mathsf{halt}} \qquad \text{for each } \sigma \in \Sigma \cup \{\sqcup\} \tag{22}$$

$$u_{\mathsf{halt}} \circ u_{\mathsf{out}} \subseteq u_{\mathsf{halt}} \tag{23}$$

Let $\Gamma_{\mathcal{M}}$ contain the constraints (1) – (23). We show that the reduction is as desired:

1. $\mathcal{M}$ reaches q^1_{fin} iff $\Gamma_{\mathcal{M}}, a \models \varphi_{\mathcal{M}}$.
   Suppose $\mathcal{M}$ reaches q^1_{fin}. Assume an arbitrary $\mathcal{I}, a$ such that $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$, and assume there is an arbitrary $d \in \Delta^{\mathcal{I}}$ such that $(a^{\mathcal{I}}, d) \in u^{\mathcal{I}}_{\mathsf{ini}}$. Since $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$ and $\mathcal{M}$ reaches q^1_{fin}, then $(a^{\mathcal{I}}, d) \in u^{\mathcal{I}}_{\mathsf{halt}}$ follows. This implies $\mathcal{I}, a \models \varphi_{\mathcal{M}}$ and hence $\Gamma, a \models \varphi_{\mathcal{M}}$. For the converse, assume $\Gamma, a \not\models \varphi_{\mathcal{M}}$. That is, there exists some $\mathcal{I}, a$ such that $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$ but $\mathcal{I}, a \not\models \varphi_{\mathcal{M}}$. That is, there is $d \in \Delta^{\mathcal{I}}$ such that $(a^{\mathcal{I}}, d) \in u^{\mathcal{I}}_{\mathsf{ini}}$ but $(a^{\mathcal{I}}, d) \notin u^{\mathcal{I}}_{\mathsf{halt}}$. Since $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$, by constraints (3), (4), (21), and (23) we have $(a^{\mathcal{I}}, d) \notin u^{\mathcal{I}}_{\mathsf{halt}}$, and $\mathcal{I}$ cannot

contain any arcs of the form $t_{q^1_{\mathsf{fin}}\sigma}$ or $f_{q^1_{\mathsf{fin}}\sigma}$, which means that the computation of $\mathcal{M}$ does not reach the state q^1_{fin}.

2. If $\mathcal{M}$ reaches q^2_{fin}, then there exists a finite $\mathcal{I}$ such that $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$ and $\mathcal{I}, a \not\models \varphi_{\mathcal{M}}$.
Suppose that there is a computation of $\mathcal{M}$ that reaches q^2_{fin}. Then this computation does not reach q^1_{fin}. Moreover, there is some $\mathcal{I}, a$ that simulates this computation, that is, such that $(a^{\mathcal{I}}, d) \in u^{\mathcal{I}}_{\mathsf{ini}}$ and $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$. Since $\mathcal{M}$ does not reach q^1_{fin}, we can assume there are no arcs $t_{q^1_{\mathsf{fin}}\sigma}$ or $f_{q^1_{\mathsf{fin}}\sigma}$ in $\mathcal{I}$, and since the only constraints that imply u_{halt} are (21) and (23), we can assume $u^{\mathcal{I}}_{\mathsf{halt}} = \emptyset$. Hence we have $\mathcal{I}, a \models \Gamma_{\mathcal{M}}$ with $(a^{\mathcal{I}}, d) \in u^{\mathcal{I}}_{\mathsf{ini}}$ but $u^{\mathcal{I}}_{\mathsf{halt}} = \emptyset$ as desired.

This concludes the proof of Theorem 14. ◀

The same proof that we gave for the pointed semantics of path constraints applies also to the global semantics, even if we further restrict the constraints to be prefix-empty.

▶ **Theorem 15.** *The problems of deciding* $\Gamma \models \varphi$ *and* $\Gamma \models_{fin} \varphi$ *given* Γ *and* φ *are undecidable, even when* φ *is of the form* $r_1 \subseteq r_2$ *and* Γ *contains only constraints of the forms* $r_1 \circ r_2 \subseteq r_3$ *and* $r_1 \subseteq r_2 \circ r_3$*, for* $\{r_1, r_2, r_3\} \subseteq \mathsf{N_R}$*.*

Proof. It suffices to observe that in the proof of Theorem 14, all constrains $[R_p](R_\ell \subseteq R_r)$ with $R_p \neq \varepsilon$ have $R_p = u_{\mathsf{in}}$, and the only role of the prefix is to ensure that $R_\ell \subseteq R_r$ fires at all nodes, and not only at a. Under global semantics, this prefix is unnecessary and we can replace each constraint $[R_p](R_\ell \subseteq R_r)$ simply by $R_\ell \subseteq R_r$. ◀

We formulated Theorems 14 and 15 for path constraint implication, but we could just as well formulate them for the problem of deciding whether a fact $r_1(a, b)$ and a set of constraints Γ that are satisfied (at a with the pointed semantics for the prefixed version, or everywhere for the prefix-empty version) imply the existence of a pair of objects in r_2, which need not be a, b (note that in this case we don't need constraint (23)). Our negative results also apply to other languages that can express Γ. For instance, the prefix-empty version of Γ can be expressed in the following restricted class of *tuple generating dependencies*:

$$r_1(x, y), r_2(y, z) \to r_3(x, z) \qquad r_1(x, z) \to \exists y. r_2(x, y), r_3(y, z)$$

Hence we obtain a proof of undecidability of (finite) entailment of a query $\exists x, y.\, r'(x, y)$, from one single fact $r(a, b)$ in the presence of dependencies in this restricted class.

We note that the undecidability of path constraint implication shows the undecidability of (finite) satisfiability in $\mathcal{ZOI}^{\setminus}$. In fact, it shows the undecidability of (finite) satisfiability of a complex $\mathcal{ZOI}^{\setminus}$ role $id(\forall R.\bot) \circ (r_1 \setminus r_2)$, where R is a union of roles of the forms

$$(r_1 \circ r_2) \setminus r_3 \qquad r_1 \setminus (r_2 \circ r_3) \qquad r \circ ((r_1 \circ r_2) \setminus r_3) \qquad r \circ (r_1 \setminus (r_2 \circ r_3)) \tag{24}$$

Undecidability also applies to the (finite) entailment of $r_2(a, b)$ from $(id(\forall R.\bot) \circ r_1)(a, b)$. If we use a set $\mathcal{K}$ of inclusions of the form $\top \sqsubseteq \forall R_\ell \setminus R_r.\bot$ (where each R_ℓ and R_r is the concatenation of at most two role names) to enforce prefix-empty constraints to hold everywhere, we get undecidability of testing whether the KB $\mathcal{K} \wedge \{r_1(a, b)\}$ (finitely) entails $r_2(a, b)$.

We remark that the constraints in the proof above can also be written in GXPath_{reg}, the extension of $\mathsf{GXPath}^{path\text{-}pos}_{reg}$ that allows for negation (and hence, intersection and difference) of path formulas. Hence Theorem 14 implies the undecidability of the (finite) implication of two GXPath_{reg} formulas ψ_1 and ψ_2, even when ψ_2 is restricted to a role name r_2 and ψ_1 is a formula of the form $[\neg\langle\xi\rangle] \circ r_1$ for r_1 a role name and ξ a union of roles of the forms in (24) above. Note that this result is tighter that the one in [27], whose proof heavily uses ε.

Improving the Upper bound for Prefix-empty Constraints. We have seen that under global semantics, or with non-empty prefixes, there is no hope for decidability of path constraint implication. However, prefix-empty constraints under the pointed semantics are expressible in plain $\mathcal{ZOI}$ without using role difference, using a nominal to denote the point at which the constraint is evaluated. Hence we can reduce their (finite) implication problem to (finite) KB satisfiability in $\mathcal{ZOI}$. This implies decidability and an ExpTime upper bound, thus significantly improving the previous 2ExpSpace bound shown in [3] for the subclass of prefix-empty one-way regular path constraints.

▶ **Theorem 16.** *Let Γ be a set of prefix-empty path constraints, φ a prefix-empty path constraint, and $a \in \mathsf{N_I}$. Then we can obtain a set of $\mathcal{ZOI}$ of axioms $\mathcal{T}_\Gamma$ and a concept C_φ such that $\Gamma \models \varphi$ iff $\{\mathcal{T}_\Gamma\} \wedge \neg C_\varphi$ is unsatisfiable. Moreover, $\mathcal{T}_\Gamma$ and C_φ can be constructed in linear time and the size of $\mathcal{T}_\Gamma$ and C_φ are both linear, in the combined sizes of Γ and φ.*

Proof. For each $\gamma = R_\ell \subseteq R_r \in \Gamma \cup \varphi$, let $\mathcal{T}_\gamma$ be the axiom $\{a\} \sqsubseteq \forall R_\ell.\exists \mathsf{inv}(R_r).\{a\}$ where $\mathsf{inv}(R_r)$ is the *inverted* R_r defined as follows, where $r \in \mathsf{N_R}$ and R, R' denote arbitrary roles:

$$\begin{array}{ccc} \mathsf{inv}(r) = r^- & \mathsf{inv}(r^-) = r & \mathsf{inv}(\{(a,b)\}) = \{(b,a)\} \\ \mathsf{inv}(id(C)) = id(C) & \mathsf{inv}(R \cap R') = \mathsf{inv}(R) \cap \mathsf{inv}(R') & \mathsf{inv}(R \cup R') = \mathsf{inv}(R) \cup \mathsf{inv}(R') \\ \mathsf{inv}(R^*) = (\mathsf{inv}(R))^* & \mathsf{inv}(R \setminus R') = \mathsf{inv}(R) \setminus \mathsf{inv}(R') & \mathsf{inv}(R \circ R') = \mathsf{inv}(R') \circ \mathsf{inv}(R) \end{array}$$

and $\mathsf{inv}(\varepsilon) = \varepsilon$. It is easy to see that $(d, d') \in R^{\mathcal{I}}$ iff $(d', d) \in \mathsf{inv}(R)^{\mathcal{I}}$ for every $d, d' \in \Delta^{\mathcal{I}}$. It is then a straightforward consequence of the semantics of $\mathcal{ZOI}$ and of γ, that for every pointed instance $\mathcal{I}, a$ we have $\mathcal{I}, a \models \gamma$ if and only if $\mathcal{I} \models \mathcal{T}_\gamma$. Hence it easily follows that $\Gamma \models \varphi$ if and only if $(\bigcup_{\gamma \in \Gamma} \mathcal{T}_\gamma) \wedge \neg \mathcal{T}_\varphi$ is unsatisfiable. ◀

From this reduction and Theorem 5 we get:

▶ **Corollary 17.** *The unrestricted and the finite pointed implication problems for prefix-empty path constraints are decidable in* ExpTime.

We can combine this result and Corollary 12, obtaining that entailment is decidable in ExpTime whenever all constraints are prefix-empty or involve only simple roles. Although this is a strong restriction, it still allows for fairly non-trivial constraints. Apart from capturing these relevant decidable subclasses of path constraints, $\mathcal{ZOI}$ can also express many other natural constraints, some of which are not easily expressible as path constraints. For example, ϕ_5 in Example 3 does not directly correspond to a path constraint. $\mathcal{ZOI}$ provides flexible means to express quite involved expressions, for example, that a course required in an undergraduate program must be taught by a faculty member that is a member of an institute, or a suborganization of it:

$$\mathsf{Course} \sqcap \exists \mathsf{requires}^-.\mathsf{UndergradProg} \sqsubseteq \exists \mathsf{teaches}^-.\big(\exists(\mathsf{memberOf} \circ \mathsf{partOf}^*).\mathsf{Inst}\big).$$

To conclude this section, we remark that both the pointed and the global semantics are directly supported within $\mathcal{ZOI}$: the logic has a global semantics, but we can use nominals to ensure that any assertion or inclusion only 'fires' at the interpretation of a given individual.

4 Verification of Evolving Graph Structured Data

We have advocated the use of $\mathcal{ZOI}$ as a constraint language for GSD. We define a language for manipulating GSD and show that it is possible to effectively reason about the preservation of $\mathcal{ZOI}$ constraints when data instances evolve as a result of operations in this language.

Updating Graph Structured Data. The basic operations in our manipulation language allow to insert or delete individuals from extensions of concepts, and pairs of individuals from extensions of roles. The candidates for additions and deletions are instances of complex concepts and roles in $\mathcal{ZOI}$. The language allows, in particular, to select one object and add it to or remove it from a concept name, using a nominal $\{a\}$. It similarly allows to add or remove pairs of objects to/from role names using a nominal role $\{(a, b)\}$. Moreover, we can add to or delete from some role or concept the answers to query expressed as a complex concept or role (in the latter case, a so-called *regular path query*). We also allow for variables in the place of individuals, to have a more natural manipulation language with parameters that can be instantiated with different values. Finally, these basic actions can be combined into complex ones using composition and conditional actions.

The language we define next is like the one in [4], but there basic actions use concepts and roles in a different DL called $\mathcal{ALCHOIQ}br$, instead of $\mathcal{ZOI}$. $\mathcal{ALCHOIQ}br$ does not allow for regular expressions as roles, hence it cannot express path queries and path constraints. On the other hand, it supports *number restrictions*, which are not allowed in $\mathcal{ZOI}$. We note that the language in [4] allows roles of the form $R|_C$ (or $\mathsf{inv}(R)|_C$), which stand for the pairs of objects in R whose first (resp., second) component is an instance of C. Such roles are expressible in $\mathcal{ZOI}$ using $R \circ id(C)$.

▶ **Definition 18** (Action language)**.** In what follows, additionally to $\mathsf{N_I}$, $\mathsf{N_C}$ and $\mathsf{N_R}$, we consider a countably infinite set $\mathsf{N_V}$ of variables, disjoint from the other sets. We use $\mathcal{ZOI}_\mathsf{V}$ concepts, roles, and KBs, which are defined as for $\mathcal{ZOI}$, but allowing for variables in the place of individuals, that is, atomic concepts take the forms A and $\{t\}$, and atomic roles the forms r, r^- and $\{(t, t')\}$, where $t, t' \in \mathsf{N_V} \cup \mathsf{N_I}$, $A \in \mathsf{N_C}$, $r \in \mathsf{N_R} \setminus \{\mathsf{T}\}$.

Basics action β and *(complex) actions* α are defined by the following grammar:

$$\beta \longrightarrow (A \oplus C) \mid (A \ominus C) \mid (r \oplus R) \mid (r \ominus R) \qquad \alpha \longrightarrow \epsilon \mid \beta \cdot \alpha \mid (\mathcal{K} \,?\, \alpha[\![\alpha]\!]) \cdot \alpha$$

with A a concept name, C an arbitrary $\mathcal{ZOI}_\mathsf{V}$ concept, r a role name, R an arbitrary $\mathcal{ZOI}_\mathsf{V}$ role, and $\mathcal{K}$ an arbitrary $\mathcal{ZOI}_\mathsf{V}$ KB. The symbol ϵ denotes the *empty action*.

We call a concept, role, KB or an action *ground* if it has no variables. A *substitution* is a function σ from $\mathsf{N_V}$ to $\mathsf{N_I}$. For a concept, role, KB or an action γ, we use $\sigma(\gamma)$ to denote the result of replacing in γ every occurrence of a variable x by the individual $\sigma(x)$. An action α' is called a *ground instance* of an action α if $\alpha' = \sigma(\alpha)$ for some substitution σ. ◀

Intuitively, an application of an action $(A \oplus C)$ on an instance $\mathcal{I}$ is the addition of the content of $C^\mathcal{I}$ to $A^\mathcal{I}$. Similarly, $(A \ominus C)$ removes the content of $C^\mathcal{I}$ from $A^\mathcal{I}$. The two operations can also be performed on roles. Composition stands for successive action execution, and a conditional action $\mathcal{K} \,?\, \alpha_1[\![\alpha_2]\!]$ expresses that α_1 is executed if the instance is a model of $\mathcal{K}$, and α_2 is executed otherwise. If $\alpha_2 = \epsilon$ then we have an action with a simple *pre-condition* as in classical planning languages, and we write it as $\mathcal{K} \,?\, \alpha_1$, omitting α_2.

To formally define the semantics of actions, we introduce the notion of *instance update*.

▶ **Definition 19** (Instance update, semantics of actions)**.** Assume an instance $\mathcal{I}$ and let E be a concept or role name. If E is a concept, let $W \subseteq \Delta^\mathcal{I}$, otherwise, if E is a role, let $W \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. Then, $\mathcal{I} \oplus_E W$ (resp., $\mathcal{I} \ominus_E W$) denotes the instance $\mathcal{I}'$ such that $\Delta^{\mathcal{I}'} = \Delta^\mathcal{I}$, $E^{\mathcal{I}'} = E^\mathcal{I} \cup W$ (resp., $E^{\mathcal{I}'} = E^\mathcal{I} \setminus W$), and $E_1^{\mathcal{I}'} = E_1^\mathcal{I}$, for all symbols $E_1 \neq E$. Given a

ground action α, we define a mapping S_α from instances to instances as follows:

$$S_\epsilon(\mathcal{I}) = \mathcal{I}$$

$$S_{(\mathcal{K}\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha}(\mathcal{I}) = \begin{cases} S_{\alpha_1\cdot\alpha}(\mathcal{I}), & \text{if } \mathcal{I} \models \mathcal{K}, \\ S_{\alpha_2\cdot\alpha}(\mathcal{I}), & \text{if } \mathcal{I} \not\models \mathcal{K}. \end{cases}$$

$$S_{(A\oplus C)\cdot\alpha}(\mathcal{I}) = S_\alpha(\mathcal{I} \oplus_A C^{\mathcal{I}})$$
$$S_{(A\ominus C)\cdot\alpha}(\mathcal{I}) = S_\alpha(\mathcal{I} \ominus_A C^{\mathcal{I}})$$
$$S_{(p\oplus r)\cdot\alpha}(\mathcal{I}) = S_\alpha(\mathcal{I} \oplus_r R^{\mathcal{I}})$$
$$S_{(p\ominus r)\cdot\alpha}(\mathcal{I}) = S_\alpha(\mathcal{I} \ominus_r R^{\mathcal{I}})$$ ◀

Note that we have not defined the semantics of actions with variables, i.e., for non-ground actions. In our approach, all variables of an action are seen as parameters whose values are given before execution by a substitution with actual individuals, i.e., by grounding.

▶ **Example 20.** Consider the next action with a free variable x, and its ground instance α_{dl}:

$$\text{RemoveCourse}(x) = \big(\mathsf{Course} \ominus \{x\}\big)\cdot\big(\mathsf{requires} \ominus (\mathsf{requires}\circ id(\{x\}))\big) \cdot\big(\mathsf{offers} \ominus (\mathsf{offers}\circ id(\{x\}))\big)\cdot\big(\mathsf{partOf} \ominus (id(\{x\})\circ \mathsf{partOf})\big)$$

$$\alpha_{dl} = \big(\mathsf{Course} \ominus \{\text{DLs:CS451}\}\big)\cdot\big(\mathsf{requires} \ominus (\mathsf{requires}\circ id(\{\text{DLs:CS451}\}))\big) \cdot\big(\mathsf{offers} \ominus (\mathsf{offers}\circ id(\{\text{DLs:CS451}\}))\big)\cdot\big(\mathsf{partOf} \ominus (id(\{\text{DLs:CS451}\})\circ \mathsf{partOf})\big)$$

In $S_{\alpha_{dl}}(\mathcal{I}_{\mathsf{Uni}})$ we have the following changes, and the rest of the instance remains unchanged:

$$\begin{aligned} \mathsf{Course}^{\mathcal{I}} &= \{\text{DataStruct:CS202}, \text{FoundDBs:CS327}\} \\ \mathsf{requires}^{\mathcal{I}} &= \{(\text{BSc\_CSci}, \text{DataStruct:CS202}), (\text{MSc\_CompLogic}, \text{FoundDBs:CS327}), \\ &\quad (\text{MSc\_CompLogic}, \text{mod\_KR})\} \\ \mathsf{offers}^{\mathcal{I}} &= \{(\text{CS\_Dept}, \text{BSc\_CSci}), (\text{CS\_Dept}, \text{MSc\_CompLogic}), \\ &\quad (\text{CS\_Dept}, \text{MSc\_Bioinformatics})\{(\text{CS\_Dept}, \text{DataStruct:CS202}), \\ &\quad (\text{CS\_Dept}, \text{FoundDBs:CS327})\} \\ \mathsf{partOf}^{\mathcal{I}} &= \{\} \end{aligned}$$

The Static Verification Problem. We consider now the scenario where DL KBs are used to impose integrity constraints on GSD. A basic reasoning problem for analyzing the effects of actions in the presence of integrity constraints is *static verification*, which consists in checking whether the execution of an action α always preserves the satisfaction of integrity constraints given by a KB $\mathcal{K}$.

▶ **Definition 21** (The static verification problem)**.** Let $\mathcal{K}$ be a KB. We say that an action α is $\mathcal{K}$*-preserving* if for every ground instance α' of α and every finite interpretation $\mathcal{I}$, we have that $\mathcal{I} \models \mathcal{K}$ implies $S_{\alpha'}(\mathcal{I}) \models \mathcal{K}$. The *static verification problem* consists on deciding, given an action α and a KB $\mathcal{K}$, whether α is $\mathcal{K}$-preserving. ◀

▶ **Example 22.** Recall the constraints $\mathcal{K}_{\mathsf{Uni}}$ from Ex. 3, and the action α_{dl} from Ex. 20. Note that α_{dl} is not $\mathcal{K}_{\mathsf{Uni}}$-preserving. In fact, this is witnessed by out instance $\mathcal{I}_{\mathsf{Uni}}$. We saw that in $S_{\alpha_{dl}}(\mathcal{I}_{\mathsf{Uni}})$ we have $(\text{MSc\_CompLogic}, \text{mod\_KR}) \in \mathsf{requires}^{\mathcal{I}}$, but $\text{mod\_KR} \notin (\exists \mathsf{partOf}^{-*}.\mathsf{Course})^{\mathcal{I}}$, that is, the mandatory KR module does not contain any courses, violating $\phi_4 = \top \sqsubseteq \forall \mathsf{requires}.(\exists \mathsf{partOf}^{-*}.\mathsf{Course})$.

Our technique for static verification relies on a transformation $\mathsf{TR}_\alpha(\mathcal{K})$ that rewrites $\mathcal{K}$ incorporating the effects of an action α. The technique is similar in spirit to *regression* in reasoning about actions [29], and it can be seen as a way to compute the *weakest precondition* of α and $\mathcal{K}$. Intuitively, the models of $\mathsf{TR}_\alpha(\mathcal{K})$ are exactly the interpretations $\mathcal{I}$ such that applying α on $\mathcal{I}$ leads to a model of $\mathcal{K}$. In this way, we can effectively reduce reasoning about changes in any database that satisfies a given $\mathcal{K}$, to reasoning about a single KB.

▶ **Definition 23.** Given a $\mathcal{ZOI}$ KB $\mathcal{K}$, we use $\mathcal{K}_{E\leftarrow E'}$ to denote the KB that is obtained from $\mathcal{K}$ by replacing every name E by the (possibly more complex) expression E'. Given a

KB $\mathcal{K}$ and a ground action α, we define $\mathsf{TR}_\alpha(\mathcal{K})$ as follows.

$$\mathsf{TR}_\varepsilon(\mathcal{K}) = \mathcal{K} \qquad \mathsf{TR}_{(r\oplus R)\cdot\alpha}(\mathcal{K}) = (\mathsf{TR}_\alpha(\mathcal{K}))_{r\leftarrow r\cup R}$$
$$\mathsf{TR}_{(A\oplus C)\cdot\alpha}(\mathcal{K}) = (\mathsf{TR}_\alpha(\mathcal{K}))_{A\leftarrow A\sqcup C} \qquad \mathsf{TR}_{(r\ominus R)\cdot\alpha}(\mathcal{K}) = (\mathsf{TR}_\alpha(\mathcal{K}))_{r\leftarrow r\setminus R}$$
$$\mathsf{TR}_{(A\ominus C)\cdot\alpha}(\mathcal{K}) = (\mathsf{TR}_\alpha(\mathcal{K}))_{A\leftarrow A\sqcap\neg C} \qquad \mathsf{TR}_{(\mathcal{K}_1\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha}(\mathcal{K}) = (\dot\neg\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_1\cdot\alpha}(\mathcal{K}))\wedge (\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_2\cdot\alpha}(\mathcal{K})).$$ ◀

Assume a ground action α and a $\mathcal{ZOI}$ KB $\mathcal{K}$. Note that the transformation $\mathsf{TR}_\alpha(\mathcal{K})$ may introduce role differences involving complex roles from α. Hence $\mathsf{TR}_\alpha(\mathcal{K})$ is a $\mathcal{ZOI}^\setminus$ and need not be a $\mathcal{ZOI}$ KB. Note also that the size of $\mathsf{TR}_\alpha(\mathcal{K})$ might be exponential in the size of α. By employing the same argument as [4], we see that the transformation correctly captures the effects of complex actions. In particular, for every interpretation $\mathcal{I}$, we have $S_\alpha(\mathcal{I}) \models \mathcal{K}$ iff $\mathcal{I} \models \mathsf{TR}_\alpha(\mathcal{K})$. With the transformation $\mathsf{TR}_\alpha(\mathcal{K})$ above we have a reduction from static verification to finite (un)satisfiability of $\mathcal{ZOI}^\setminus$ KBs: an action α is not $\mathcal{K}$-preserving iff some finite model of $\mathcal{K}$ does not satisfy $\mathsf{TR}_{\alpha^*}(\mathcal{K})$, where α^* is a 'canonical' grounding of α. Formally, we have:

▶ **Theorem 24** ([4]). *For a (complex) action α and a KB $\mathcal{K}$, the following are equivalent:*

(i) *The action α is not $\mathcal{K}$-preserving.*

(ii) *$\mathcal{K} \wedge \dot\neg\mathsf{TR}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable, where α^* is obtained from α by replacing each variable with a fresh individual name not occurring in α and $\mathcal{K}$.*

Undecidability of unrestricted static verification. The first and foremost consequence of this reduction is that for the action language we have defined, the static verification problem is undecidable, even if the input $\mathcal{K}$ is trivial, and the actions are quite restricted:

▶ **Theorem 25.** *Deciding whether α is $\mathcal{K}$-preserving is undecidable, even when $\mathcal{K}$ is a trivial KB of the form $r(a,b)$, and α is just a sequence of basic actions of the forms $(r \oplus R)$ and $(r \ominus R)$, with R a sequence of one or two concatenated role names.*

Proof. We provide a reduction from deciding $\Gamma \models_{fin} r_1 \subseteq r_2$, where Γ contains only constraints of the forms $r_1 \circ r_2 \subseteq r_3$ and $r_1 \subseteq r_2 \circ r_3$ for $\{r_1, r_2, r_3\} \subseteq \mathsf{N_R}$. We have seen above that this problem is undecidable. In particular, we construct an action α such that $\Gamma \models_{fin} r_1 \subseteq r_2$ iff α is $r_1(a,b)$-preserving. Let $R_1^1 \subseteq R_2^1, \ldots, R_1^n \subseteq R_2^n$ be an enumeration of all constraints in Γ. For every $1 \leq i \leq n$, let p_1^i and p_2^i be fresh role names. Then α is the concatenation of the following actions in the given order:

- $(r_1 \ominus r_1) \cdot (r_1 \oplus r_2)$
- $(p_1^1 \ominus p_1^1) \cdot \cdots \cdot (p_1^n \ominus p_1^n) \cdot (p_2^1 \ominus p_2^1) \cdot \cdots \cdot (p_2^n \ominus p_2^n)$
- $(p_1^1 \oplus R_1^1) \cdot \cdots \cdot (p_1^n \oplus R_1^n) \cdot (p_2^1 \oplus R_2^1) \cdot \cdots \cdot (p_2^n \oplus R_2^n)$
- $(p_1^1 \ominus p_2^1) \cdot \cdots \cdot (p_1^n \ominus p_2^n) \cdot (r_1 \oplus p_1^1) \cdot \cdots \cdot (r_1 \oplus p_1^n)$.

Recall that α is not $r_1(a,b)$-preserving iff $r_1(a,b) \wedge \neg\mathsf{TR}_\alpha(r_1(a,b))$ is finitely satisfiable. It's not hard to see that $\mathsf{TR}_\alpha(r_1(a,b)) = R_\Gamma(a,b)$, where R_Γ is equivalent to the role $r_2 \cup (R_1^1 \setminus R_2^1) \cup \cdots \cup (R_1^n \setminus R_2^n)$. Thus $r_1(a,b) \wedge \neg\mathsf{TR}_\alpha(r_1(a,b))$ is finitely satisfiable iff $\Gamma \not\models_{fin} r_1 \subseteq r_2$. ◀

Unfortunately we cannot allow for complex roles in our actions, not even of the form $r \circ r'$, but we get positive results if we restrict actions to *simple* roles.

An undesired effect of disallowing complex roles in actions is that we cannot express $r|_C$ as $r \circ id(C)$, and as our examples illustrate, this construct is quite useful. For nominals we can, however, simulate $r|_{\{a\}}$ in $\mathcal{ZOI}$. We use a special role name $\mathsf{T}|_{\{a\}}$ with the intended

semantics $(\mathsf{T}|_{\{a\}})^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \{a\}^{\mathcal{I}}$, and write the simple role $r \cap \mathsf{T}|_{\{a\}}$ in the place of $r|_{\{a\}}$. The intended meaning of $\mathsf{T}|_{\{a\}}$ is easily enforced by adding $\top \sqsubseteq \exists \mathsf{T}|_{\{a\}}.a$ to any $\mathcal{ZOI}$ KB.

We call an action α *role-restricted* if in every basic action of the form $(r \oplus R)$ or $(r \ominus R)$, we have that R is a simple $\mathcal{ZOI}_{\mathsf{V}}$ role that may use the special role names $\mathsf{T}|_{\{a\}}$.

Note α_{dl} in Example 20 can be rewritten as a role-restricted action as follows:

$$\alpha'_{dl} = \big(\mathsf{Course} \ominus \{\mathsf{DLs{:}CS451}\}\big) \cdot \big(\mathsf{requires} \ominus (\mathsf{requires} \cap \mathsf{T}|_{\{\mathsf{DLs:CS451}\}})\big) \cdot \big(\mathsf{offers} \ominus (\mathsf{offers} \cap \mathsf{T}|_{\{\mathsf{DLs:CS451}\}})\big) \cdot \big(\mathsf{partOf} \ominus (\mathsf{partOf} \cap (\mathsf{T}|^{-}_{\{\mathsf{DLs:CS451}\}}))\big).$$

▶ **Theorem 26.** *Deciding whether α is $\mathcal{K}$-preserving for a given $\mathcal{ZOI}$ KB $\mathcal{K}$ and a role-restricted α is* ExpTime*-complete.*

Proof sketch. Since the union and the difference of simple roles are simple roles, it is not hard to see that the result of iteratively replacing role names by simple roles involving union and difference in a $\mathcal{ZOI}$ role results in a $\mathcal{ZOI}$ role. Hence, for any $\mathcal{ZOI}$ KB $\mathcal{K}$ and a role-restricted action α, the KB $\mathsf{TR}_{\alpha}(\mathcal{K})$ is not only a $\mathcal{ZOI}^{\setminus}$ KB but also a $\mathcal{ZOI}$ KB (i.e., difference is applied to simple roles only). Then from the decidability of (finite) satisfiability of $\mathcal{ZOI}$ it follows that checking whether α is $\mathcal{K}$-preserving is decidable. For the complexity upper bound, recall that the size of $\mathsf{TR}_{\alpha}(\mathcal{K})$ might be exponential in the size of α. However, as argued in [4], there are only exponentially many conjunctive clauses in disjunctive normal from of $\mathcal{K} \wedge \dot{\neg}\mathsf{TR}_{\alpha^*}(\mathcal{K})$, each with size polynomial in the size of α and $\mathcal{K}$. Thus from Theorems 5 and 24 we obtain the desired result. ◀

5 Conclusions and Outlook

The main goal of this work was to advocate the use of the DL $\mathcal{ZOI}$ to specify constraints over graph structured data and to show the decidability of static verification in a rich language for manipulating such data. Along the way, we have shown several undecidability and complexity results that concern not only our setting, but also formalisms that were introduced in the 90s, as well as recently introduced query languages for GSD like GXPath_{reg}. In our future work we aim at providing some support for identification constraints, which is clearly desirable but naturally requires equality reasoning. This is challenging, as e.g., the decidability of the extension of $\mathcal{ZOI}$ where some roles must be interpreted as partial functions is a long standing open problem.

Acknowledgements. This work was supported by the EU IP project Optique (*Scalable End-user Access to Big Data*), grant agreement n. FP7-318338, the Vienna Science and Technology Fund project ICT12-15, and the Austrian Science Fund projects P25207 and T515.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

2 Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

3 Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *J. of Computer and System Sciences*, 58(3):428–452, 1999.

4 Shqiponja Ahmetaj, Diego Calvanese, Magdalena Ortiz, and Mantas Šimkus. Managing change in Graph-structured Data using Description Logics. In *Proc. of the 28th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 966–973. AAAI Press, 2014.

5 Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys*, 40(1):1:1–1:39, 2008. doi:10.1145/1322432.1322433.

6 Marcelo Arenas, Wenfei Fan, and Leonid Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. on Computing*, 38(3):841–880, 2008. doi:10.1137/050646895.

7 Alessandro Artale, Diego Calvanese, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev. Reasoning over extended ER models. In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER)*, volume 4801 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2007.

8 Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

9 Jie Bao et al. OWL 2 Web Ontology Language document overview (second edition). W3C Recommendation, World Wide Web Consortium, December 2012. URL: http://www.w3.org/TR/owl2-overview/.

10 Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.

11 Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web. Web Logic Rules – 11th Int. Summer School Tutorial Lectures (RW)*, volume 9203 of *Lecture Notes in Computer Science*, pages 218–307. Springer, 2015. doi:10.1007/978-3-319-21768-0_9.

12 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

13 Peter Buneman. Semistructured data. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 117–121, 1997.

14 Peter Buneman, Wenfei Fan, and Scott Weinstein. Path constraints in semistructured databases. *J. of Computer and System Sciences*, 61(2):146–193, 2000. doi:10.1006/jcss.2000.1710.

15 Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment and answering under description logics constraints. *ACM Trans. on Computational Logic*, 9(3):22.1–22.31, 2008.

16 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

17 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. *Theoretical Computer Science*, 371(3):169–182, 2007.

18 Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Regular path queries in expressive description logics with nominals. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 714–720, 2009.

19 Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov. Evolution of *DL-Lite* knowledge bases. In *Proc. of the 9th Int. Semantic Web Conf. (ISWC)*, volume 6496 of *Lecture Notes in Computer Science*, pages 112–128. Springer, 2010.

20 Diego Calvanese, Magdalena Ortiz, and Mantas Šimkus. Containment of regular path queries under description logic constraints. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 805–812, 2011.

21 Giuseppe De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.

22 Alin Deutsch and Val Tannen. Optimization properties for classes of conjunctive regular path queries. In *Proc. of the 8th Int. Workshop on Database Programming Languages (DBPL)*, volume 2397 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2001.

23 Wenfei Fan and Leonid Libkin. On XML integrity constraints in the presence of DTDs. *J. of the ACM*, 49(3):368–406, 2002.

24 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.

25 Erich Grädel and Martin Otto. On logics with two variables. *Theoretical Computer Science*, 224:73–113, 1999.

26 Gösta Grahne and Alex Thomo. Query containment and rewriting using views for regular path queries under constraints. In *Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 111–122, 2003.

27 Egor V. Kostylev, Juan L. Reutter, and Domagoj Vrgoc. Containment of data graph queries. In *Proc. of the 17th Int. Conf. on Database Theory (ICDT)*, pages 131–142. OpenProceedings.org, 2014. doi:10.5441/002/icdt.2014.16.

28 Maurizio Lenzerini. Ontology-based data management. In *Proc. of the 20th Int. Conf. on Information and Knowledge Management (CIKM)*, pages 5–6, 2011. doi:10.1145/2063576.2063582.

29 H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming*, 31:59–84, 1997.

30 Leonid Libkin, Wim Martens, and Domagoj Vrgoc. Querying graph databases with XPath. In *Proc. of the 16th Int. Conf. on Database Theory (ICDT)*, pages 129–140. ACMP, 2013. doi:10.1145/2448496.2448513.

31 Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Updating description logic ABoxes. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 46–56, 2006.

32 Michael Mortimer. On languages with two variables. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:135–140, 1975.

33 Magdalena Ortiz. Ontology based query answering: The story so far. In *Proc. of the 7th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW)*, volume 1087 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/, 2013.

34 Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008. doi:10.1007/978-3-540-77688-8_5.

35 Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 466–471, 1991.

36 Peter T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012. doi:10.1145/2206869.2206879.

Query Stability in Monotonic Data-Aware Business Processes

Ognjen Savković[1], Elisa Marengo[2], and Werner Nutt[3]

1 **Free University of Bozen-Bolzano, Bolzano, Italy**
`ognjen.savkovic@unibz.it`
2 **Free University of Bozen-Bolzano, Bolzano, Italy**
`elisa.marengo@unibz.it`
3 **Free University of Bozen-Bolzano, Bolzano, Italy**
`werner.nutt@unibz.it`

Abstract

Organizations continuously accumulate data, often according to some business processes. If one poses a query over such data for decision support, it is important to know whether the query is *stable,* that is, whether the answers will stay the same or may change in the future because business processes may add further data. We investigate query stability for conjunctive queries. To this end, we define a formalism that combines an explicit representation of the control flow of a process with a specification of how data is read and inserted into the database. We consider different restrictions of the process model and the state of the system, such as negation in conditions, cyclic executions, read access to written data, presence of pending process instances, and the possibility to start fresh process instances. We identify for which restriction combinations stability of conjunctive queries is decidable and provide encodings into variants of Datalog that are optimal with respect to the worst-case complexity of the problem.

1998 ACM Subject Classification H.2.4 [Systems]: Relational databases

Keywords and phrases Business Processes, Query Stability

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.16

1 Introduction

Data quality focuses on understanding how much data is fit for its intended use. This problem has been investigated in database theory, considering aspects such as consistency, currency, and completeness [8, 13, 23]. A question that these approaches consider only marginally is where data originates and how it evolves.

Although in general a database may evolve in arbitrary ways, often data are generated according to some business process, implemented in an information system that accesses the DB. We believe that analyzing how business processes generate data allows one to gather additional information on their fitness for use. In this work, we focus on a particular aspect of data quality, that is the problem whether a business process that reads from and writes into a database can affect the answer of a query or whether the answer will not change as a result of the process. We refer to this problem as query stability.

For example, consider a student registration process at a university. The university maintains a relation *Active*(*course*) with all active courses and a table *Registered*(*student*, *course*) that records which students have been registered for which course. Suppose we have a process model that does not allow processes to write into *Active* and which states that before a student is registered for a course, there must be a check that the course is active.

Consider the query Q_{agro} that asks for all students registered for the MSc in Agronomics (*mscAgro*). If *mscAgro* does not occur in *Active*, then no student can be registered and the query is stable. Consider next the query $Q_{courses}$ that asks for all courses for which some student is registered. If for each active course there is at least one student registered, then again the query is stable, otherwise, it is not stable because some student could register for a so far empty active course.

In general, query results can be affected by the activities of processes in several ways. Processes may store data from outside in the database, e.g., the application details submitted by students are stored in the database. Processes may not proceed because data does not satisfy a required condition, e.g., an applicant cannot register because his degree is not among the recognized degrees. Processes may copy data from one part of a database to another one, e.g., students who passed all exams are automatically registered for the next year. Processes may interact with each other in that one process writes data that is read by another one, e.g., the grades of entry exams stored by the student office are used by academic admission committees. Finally, some activities depend on deadlines so that data cannot change before or after a deadline.

Approach. Assessing query stability by leveraging on processes gives rise to several research questions.

1. What is a good model to represent processes, data and the interplay among the two?
2. How can one reason on query stability in such a model and how feasible is that?
3. What characteristics of the model may complicate reasoning?

(1) Monotonic Data-Aware Business Process Model. Business processes are often specified in standardized languages, such as BPMN [22], and organizations rely on engines that can run those processes (e.g., Bonita [7], Bizagi [16]). However, in these systems how the data is manipulated by the process is implicit in the code. Current theory approaches either focus on *process* modeling, representing the data in a limited way (like in Petri Nets [18]), or adopt a *data* perspective, leaving the representation of the process flow implicit [6, 4, 11]. We introduce a formalism called Monotonic Data-aware Business Processes (MDBPs). In MDBPs the process is represented as a graph. The interactions with an underlying database are expressed by annotating the graph with information on which data is read from the database and which is written into it. In MDBPs it is possible that several process instances execute the process. New information (fresh data) can be brought into the process by starting a fresh process instance (Section 2). MDBPs are monotonic in that data can only be inserted, but not deleted or updated.

(2) Datalog Encodings. Existing approaches aim at the verification of general (e.g. temporal) properties, for which reasoning is typically intractable [4, 10, 11]. In contrast, we study a specific property, namely stability of conjunctive queries (Section 3), over processes that only insert data. This allows us to map the problem to the one of query answering in Datalog. The encoding generates all maximal representative extensions of the database that can be produced in the process executions and checks if any new query answer is produced. We prove that our approach is optimal w.r.t. worst case complexity in the size of the data, query, process model and in the size of the entire input.

(3) MDBP Variants. When modeling processes and data, checking properties often becomes highly complex or undecidable. While other approaches in database theory aim at exploring

the frontiers of decidability by restricting the possibility to introduce fresh data, we adopt a more bottom-up approach and focus on a simpler problem that can be approached by established database techniques. To understand the sources of complexity of our reasoning problem, we identify *five restrictions* of MDBPs:

(i) negation is (is not) allowed in process conditions;
(ii) the process can (cannot) start with pending instances;
(iii) a process can (cannot) have cycles;
(iv) a process can (cannot) read from relations that it can write;
(v) new instances can (cannot) start at any moment.

We investigate the stability problem for each combination of the restrictions above, called variants (Sections 3–9).

Related work and conclusions end the paper (Sections 10, 11). A technical report, with complete encodings and proofs can be found in [24].

A preliminary version of this paper was presented at the AMW workshop [21].

2 Monotonic Data-Aware Business Processes

Monotonic Data-aware Business Processes (MDBPs) are the formalism by which we represent business processes and the way they manipulate data. We rely on this formalism to perform reasoning on query stability.

Notation. We adopt standard notation from databases. In particular, we assume an infinite set of relation symbols, an infinite set of constants dom as the *domain of values*, and the positive rationals $\mathbb{Q}^+$ as the *domain of timestamps*. A schema is a finite set of relation symbols. A *database instance* is a finite set of ground atoms, called *facts*, over a schema and the *domain* $dom_{\mathbb{Q}^+} = dom \cup \mathbb{Q}^+$.We use upper-case letters for variables, lower-case for constants, and overline for tuples, e.g., $\bar{c}$.

An MDBP is a pair $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$, consisting of a *process model* $\mathcal{P}$ and a *configuration* $\mathcal{C}$. The process model defines how and under which conditions actions change data stored in the configuration. The configuration is dynamic, consisting of

(i) a database,and
(ii) the process instances.

Process Model. The process model is a pair $\mathcal{P} = \langle N, L \rangle$, comprising a directed multi-graph N, the *process net*, and a *labeling function* L, defined on the edges of N.

The net $N = \langle P, T \rangle$ consists of a set of vertices P, the *places*, and a multiset of edges T, the *transitions*. A process instance traverses the net, starting from the distinguished place *start*. The transitions emanating from a place represent alternative developments of an instance.

A process instance has input data associated with it, which are represented by a fact $In(\bar{c}, \tau)$, where In is distinguished relation symbol, $\bar{c}$ is a tuple of constants from $dom_{\mathbb{Q}^+}$, and $\tau \in \mathbb{Q}^+$ is a time stamp that records when the process instance was started. We denote with $\Sigma_{\mathcal{B},In}$ and $\Sigma_{\mathcal{B}}$ the schemas of $\mathcal{B}$ with and without In, respectively.

The labeling function L assigns to every transition $t \in T$ a pair $L(t) = (E_t, W_t)$. Here, E_t, the *execution condition,* is a Boolean query over $\Sigma_{\mathcal{B},In}$ and W_t, the *writing rule,* is a rule $R(\bar{u}) \leftarrow B_t(\bar{u})$ whose head is a relation of $\Sigma_{\mathcal{B}}$ and whose body is a $\Sigma_{\mathcal{B},In}$-query that has the same arity as the head relation. Evaluating W_t over a $\Sigma_{\mathcal{B},In}$-instance $\mathcal{D}$ results in the set of facts $W_t(\mathcal{D}) = \{R(\bar{c}) \mid \bar{c} \in B_t(\mathcal{D})\}$. Intuitively, E_t specifies in which state of the database

which process instance can perform the transition t, and W_t specifies which new information is (or can be) written into the database when performing t. In this paper we assume that E_t and B_t are conjunctive queries, possibly with negated atoms and inequality atoms with "$<$" and "$\leq$" involving timestamps. We assume inequalities to consist of one constant and one variable, like $X < 1^{\text{st}}$ Sep. We introduce these restricted inequalities so that we can model deadlines, without introducing an additional source of complexity for reasoning.

Configuration. This component models the dynamics of an MDBP. Formally, a configuration is a triple $\langle \mathcal{I}, \mathcal{D}, \tau \rangle$, consisting of a part $\mathcal{I}$ that captures the process instances, a database instance $\mathcal{D}$ over $\Sigma_{\mathcal{B}}$, and a timestamp τ, the current time. The instance part, again, is a triple $\mathcal{I} = \langle O, M_{In}, M_P \rangle$, where $O = \{o_1, ..., o_k\}$ is a set of objects, called *process instances*, and M_{In}, M_P are mappings, associating each $o \in O$ with a fact $M_{In}(o) = In(\bar{c}, \tau)$, its input record, and a place $M_P(o) \in P$, its current, respectively.

The input record is created when the instance starts and cannot be changed later on. While the data of the input record may be different from the constants in the database, they can be copied into the database by writing rules. A process instance can see the entire database, but only its own input record.

For convenience, we also use the notation $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D}, \tau \rangle$, $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ (when τ is not relevant), or $\mathcal{B} = \langle \mathcal{P}, \mathcal{D} \rangle$ (for a process that is initially without running instances).

Execution of an MDBP. Let $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$ be an MDBP, with current configuration $\mathcal{C} = \langle \mathcal{I}, \mathcal{D}, \tau \rangle$. There are two kinds of *atomic execution* steps of an MDBP:

(i) the *traversal* of a transition by an instance and
(ii) the *start* of a new instance.

(i) Traversal of a transition. Consider an instance $o \in O$ with record $M_{In}(o) = In(\bar{c}, \tau')$, currently at place $M_P(o) = q$. Let t be a transition from q to p, with execution condition E_t. Then t is *enabled for o,* i.e., o can *traverse* t, if E_t evaluates to *true* over the database $\mathcal{D} \cup \{In(\bar{c}, \tau')\}$. Let $W_t \colon R(\bar{u}) \leftarrow B_t(\bar{u})$ be the writing rule of t. Then the effect of o traversing t is the transition from $\mathcal{C} = \langle \mathcal{I}, \mathcal{D}, \tau \rangle$ to a new configuration $\mathcal{C}' = \langle \mathcal{I}', \mathcal{D}', \tau \rangle$, such that

(i) the set of instances O and the current time τ are the same;
(ii) the new database instance is $\mathcal{D}' = \mathcal{D} \cup W_t(\mathcal{D} \cup \{In(\bar{c}, \tau')\})$, and
(iii) $\mathcal{I} = \langle O, M_{In}, M_P \rangle$ is updated to $\mathcal{I}' = \langle O, M_{In}, M'_P \rangle$ reflecting the change of place for the instance o, that is $M'_P(o) = p$ and $M'_P(o') = M_P(o')$ for all other instances o'.

(ii) Start of a new instance. Let o' be a fresh instance and let $In(\bar{c}', \tau')$ be an In-fact with $\tau' \geq \tau$, the current time of $\mathcal{C}$. The result of starting o' with info $\bar{c}'$ at time τ' is the configuration $\mathcal{C}' = \langle \mathcal{I}', \mathcal{D}, \tau' \rangle$ where $\mathcal{I}' = \langle O', M'_{In}, M'_P \rangle$ such that

(i) the database instance is the same as in $\mathcal{C}$,
(ii) the set of instances $O' = O \cup \{o'\}$ is augmented by o', and
(iii) the mappings M'_{In} and M'_P are extensions of M_{In} and M_P, resp., obtained by defining $M'_{In}(o') = In(\bar{c}', \tau')$ and $M'_P(o') = start$.

An *execution* Υ of $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$ is a finite sequence of configurations $\mathcal{C}_1, \ldots, \mathcal{C}_n$

(i) starting with $\mathcal{C}$ $(= \mathcal{C}_1)$, where
(ii) each $\mathcal{C}_{i+1}$ is obtained from $\mathcal{C}_i$ by an atomic execution step.

We denote Υ also with $\mathcal{C}_1 \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_n$. We say that the execution Υ *produces* the facts $A_1, \ldots, A_n$ if the database of the last configuration $\mathcal{C}_n$ in Υ contains $A_1, \ldots, A_n$. Since at each step a new instance can start, or an instance can write new data,

Table 1 Computational complexity of query stability in MDBPs. The results in a row hold for the class of MDBPs satisfying the defining restrictions and for the subclasses satisfying one or more of the optional restrictions. The results for all decidable variants indicate matching lower and upper bounds (except for AC^0). The $*$ indicates that the results for rowo hold for all non-trivial combinations of restrictions. All results for data, process, query and combined complexity of the decidable variants hold already for singleton MDBPs. †Note that, in all fresh variants instance complexity can be trivially decided in constant time (omitted in the table).

Defining Restrictions	Optional Restrictions	Data	Instance	Process	Query	Combined	Sect.
—	*fresh*†,*acyclic*	Undec.	Undec.	Undec.	Undec.	Undec.	4
closed	—	co-NP	co-NP	co-NExpTime	Π_2^p	co-NExpTime	6
positive	*closed*	PTime	co-NP	ExpTime	Π_2^p	ExpTime	5, 8
positive	*fresh*†, *acyclic*	PTime	co-NP	ExpTime	Π_2^p	ExpTime	8
closed, acyclic	*positive*	in AC^0	co-NP	PSpace	Π_2^p	PSpace	7
rowo	$*^\dagger$	in AC^0	in AC^0	co-NP	Π_2^p	Π_2^p	9

(i) there are infinitely many possible executions, and
(ii) the database may grow in an unbounded way over time.

3 The Query Stability Problem

In this section, we define the problem of query stability in MDBPs with its variants.

▶ **Definition 1** (Query Stability). Given $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$ with database instance $\mathcal{D}$, a query Q, and a timestamp τ, we say that Q *is stable in* $\mathcal{B}$ *until* τ, if for every execution $\mathcal{C} \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}'$ in $\mathcal{B}$, where $\mathcal{C}'$ has database $\mathcal{D}'$ and timestamp τ' such that $\tau' < \tau$, it holds that

$$Q(\mathcal{D}) = Q(\mathcal{D}').$$

If the query Q is stable until time point ∞, we say it is *globally stable*, or simply, *stable*.

The interesting question from an application view is: *Given an MDBP* $\mathcal{B}$*, a query* Q*, and a timestamp* τ*, is* Q *stable in* $\mathcal{B}$ *until* τ*?* Stability until a time-point τ can be reduced to global stability. One can modify a given MDBP by adding a new *start* place and connecting it to the old *start* place via a transition that is enabled only for instances with timestamp smaller than τ. Then a query Q is globally stable in the resulting MDBP iff in the original MDBP it is stable until τ.

To investigate sources of complexity and provide suitable encodings into Datalog, we identify five restrictions on MDBPs.

▶ **Definition 2** (Restriction on MDBPs and MDBP Executions). Let $\mathcal{B}$ be an MDBP.

Positive: $\mathcal{B}$ is *positive* if execution conditions and writing rules contain only positive atoms;
Fresh: $\mathcal{B}$ is *fresh* if its configuration does not contain any running instances;
Acyclic: $\mathcal{B}$ is *acyclic* if the process net is cycle-free;
Rowo: $\mathcal{B}$ is *rowo* (= read-only-write-only) if the schema Σ of $\mathcal{B}$ can be split into two disjoint schemas: the reading schema Σ_r and the writing schema Σ_w, such that execution conditions and queries in the writing rules range over Σ_r while the heads range over Σ_w;

Closed: an execution of $\mathcal{B}$ is *closed* if it contains only transition traversals and no new instances are started.

We will develop methods for stability checking in MDBPs for all combinations of those five restrictions. For convenience, we will say that an MDBP $\mathcal{B}$ is *closed* if we consider only closed executions of $\mathcal{B}$. A *singleton MDBP* is a closed MDBP with a single instance in the initial configuration.

Complexity Measures. The input for our decision problem are an MDBP $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$, consisting of a process model $\mathcal{P}$, an instance part $\mathcal{I}$, a database $\mathcal{D}$ and a timestamp τ, and a query Q. The question is: *Is Q globally stable in $\langle \mathcal{P}, \mathcal{I}, \mathcal{D}, \tau \rangle$?* We refer to process, instance, data, and query complexity if all parameters are fixed, except the process model, the instance part, the database, or the query, respectively.

Roadmap. As a summary of our results, Table 1 presents the complexity of the possible variants of query stability. Each section of the sequel will cover one row.

Datalog Notation. We assume familiarity with Datalog concepts such as *least fixpoint* and *stable model* semantics, and *query answering* over Datalog programs under both semantics. We consider Datalog programs that are *recursive*, *non-recursive*, *positive*, *semipositive*, *with negation*, or *with stratified negation* [9]. We write $\Pi \cup \mathcal{D}$ to denote a program where Π is a set of rules and $\mathcal{D}$ is a set of facts.

4 Undecidable MDBPs

With negation in execution conditions and writing rules, we can create MDBPs that simulate Turing machines (TMs). Consequently, in the general variant query stability is undecidable.

Due to lack of space we only provide an intuition. To show undecidability in data complexity, we define a database schema that allows us to store a TM and we construct a process model that simulates the executions of the stored TM. MDBPs cannot update facts in the database. However, we can augment relations with an additional version argument and simulate updates by adding new versions of facts. Exploiting negation in conditions and rules we can then refer to the last version of a fact. To simulate the TM execution, the process model uses fresh constants to model (i) an unbounded number of updates of the TM configurations (= number of execution steps in the TM), and (ii) a potentially infinite tape. The TM halts iff the process produces the predicate *dummy*. Undecidability in process complexity follows from undecidability in data complexity, since a process can first write the encoding of the TM into an initially empty database. Similarly, we obtain undecidability in instance complexity using instances that write the encoding of the TM at the beginning. To obtain undecidability in query complexity we extend the encoding for data complexity such that the database encodes a universal TM and an input of the TM is encoded in the query.

▶ **Theorem 3** (Undecidability). *Query stability in MDBPs is undecidable in data, process and query complexity. It is also undecidable in instance complexity except for fresh variants for which it is constant. Undecidability already holds for acyclic MDBPs.*

In our reduction it is the unbounded number of fresh instances that are causing writing rules to be executed an unbounded number of times, so that neither cycles nor existing instances are contributing to undecidability. In the sequel we study MDBPs that are positive, closed, or rowo, and show that in all three variants stability is decidable.

5 Positive Closed MDBPs

In cyclic positive MDBPs, executions can be arbitrarily long. Still, in the absence of fresh instances, it is enough to consider executions of bounded length to check stability. Consider a positive MDBP $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$, possibly with cycles and disallowing fresh instances to start, with c different constants, r relations, k running instances, m transitions and a as the maximal arity of a relation in $\mathcal{P}$. We observe:

(i) For each relation R in $\mathcal{P}$ there are up to $c^{arity(R)}$ new R-facts that $\mathcal{B}$ can produce. Thus, $\mathcal{B}$ can produce up to rc^a new facts in total.

(ii) It is sufficient to consider executions that produce at least one new fact each mk steps. An execution that produces no new facts in mk steps has at least one instance that in those mk steps visits the same place twice without producing a new fact; those steps can be canceled without affecting the facts that are produced.

(iii) Hence, it is sufficient to consider executions of maximal length $mkrc^a$.

Among these finitely many executions, it is enough to consider those that produce a maximal set of new facts. Since a process instance may have the choice among several transitions, there may be several such maximal sets. We identify a class of executions in positive closed MDBPs, called *greedy executions*, that produce all maximal sets.

Greedy Executions. Intuitively, in a greedy execution instances traverse all cycles in the net in all possible ways and produce all that can be produced before leaving the cycle. To formalize this idea we identify two kinds of execution steps: *safe steps* and *critical steps*. A safe step is an execution step of an instance after which, given the current state of the database, the instance can return to its original place. A critical step is an execution step that is not safe. Based on this, we define *greedy sequences* and *greedy executions*. A greedy sequence is a sequence of safe steps that produces the largest number of new facts possible. A greedy execution is an execution where greedy sequences and critical steps alternate.

Let Υ be a greedy execution with i alternations of greedy sequences and critical steps. In the following, we characterize which are the transitions that instances traverse in the $i+1$-th greedy sequence and then in the $i+1$-th critical step. For a process instance o and the database D_Υ produced after Υ we define the *enabled graph* $N_{\Upsilon,o}$ as the multigraph whose vertices are the places of N (i.e., the process net of $\mathcal{B}$) and edges those transitions of N that are enabled for o given database D_Υ. Let $SCC(N_{\Upsilon,o})$ denote the set of strongly connected components (SCCs) of $N_{\Upsilon,o}$. Note that two different instances may have different enabled graphs and thus different SCCs. For a place p, let $N^p_{\Upsilon,o}$ be the SCC in $SCC(N_{\Upsilon,o})$ that contains p. Suppose that o is at place p after Υ. Then in the next greedy sequence, each instance o traverses the component $N^p_{\Upsilon,o}$ in all possible ways until no new facts can be produced, meaning that all instances traverse in an arbitrary order. Conversely, the next critical step is an execution step where an instance o traverses a transition that is not part of $N^p_{\Upsilon,o}$, and thus it leaves the current SCC. We observe that when performing safe transitions new facts may be written and new transitions may become executable. This can make SCCs of $N_{\Upsilon,o}$ to grow and merge, enabling new safe steps. With slight abuse of notation we denote such maximally expanded SCCs with $N_{\Upsilon,o}$, and with $N^p_{\Upsilon,o}$ the maximal component that contains p.

Properties of Greedy Executions. We identify three main properties of greedy executions.

- A greedy execution is characterized by its critical steps, because an instance may have to choose one among several possible critical steps. In contrast, how safe steps compose a

greedy sequence is not important for stability because all greedy sequences produce the same (maximal) set of facts.

- A greedy execution in an MDBP with m transitions and k instances can have at most mk critical steps. The reason is that an execution step can be critical only the first time it is executed, and any time after that it will be a safe step.
- Each execution can be transformed into a greedy execution such that if a query is instable in the original version then it is instable also in the greedy version. In fact, an arbitrary execution has at most mk critical steps. One can construct a greedy version starting from those critical steps, such that the other steps are part of the greedy sequences.

▶ **Lemma 4.** *For each closed execution Υ in a positive MDBP $\mathcal{B}$ that produces the set of ground atoms W, there exists a greedy execution Υ' in $\mathcal{B}$ that also produces W.*

Therefore, to check stability it is enough to check stability over greedy executions. In the following we define Datalog rules that compute facts produced by greedy executions.

Encoding into Datalog. Let $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ be a positive MDBP with m transitions and k instances. Since critical steps uniquely characterize a greedy execution, we use a tuple of size up to mk to encode them. For example, if in a greedy execution Υ at the first critical step instance o_{l_1} traverses transition t_{h_1}, in the second o_{l_2} traverses t_{h_2}, and so on up to step i, we encode this with the tuple

$$\bar{\omega} = \langle o_{l_1}, t_{h_1}, \ldots, o_{l_i}, t_{h_i} \rangle.$$

Next, we define the relations used in the encoding.

(i) For each relation R in $\mathcal{P}$ we introduce relations R^i (for i up to mk) to store all R-facts produced by an execution with i critical steps. Let Υ be the execution from above and let $\langle o_{l_1}, t_{h_1}, \ldots, o_{l_i}, t_{h_i} \rangle$ be the tuple representing it. Then, a fact of relation R^i has the form $R^i(o_{l_1}, t_{h_1}, \ldots, o_{l_i}, t_{h_i}; \bar{s})$, and it holds iff Υ produces the fact $R(\bar{s})$. Later on we use $\bar{\omega}$ to represent the tuple $\langle o_{l_1}, t_{h_1}, \ldots \rangle$. Facts of R^i are then represented as $R^i(\bar{\omega}; \bar{s})$. For convenience, we use a semicolon (;) instead of a comma (,) to separate encodings of different types in the arguments.

(ii) To record the positions of instances after each critical step we introduce relations $State^i$ such that $State^i(\bar{\omega}; p_1, \ldots, p_k)$ encodes that after Υ is executed, instance o_1 is at p_1, o_2 is at p_2, and so on.

(iii) To store the SCCs of the enabled graph we introduce relations SCC^i such that for a process instance o and a place p, the transition t belongs to $N^p_{\Upsilon,o}$ iff $SCC^i(\bar{\omega}; o, p, t)$ is true.

(iv) To compute the relations SCC^i, we first need to compute which places are reachable by an instance o from place p. For that we introduce auxiliary relations $Reach^i$ such that in the enabled graph $N_{\Upsilon,o}$ instance o can reach place p' from p iff $Reach^i(\bar{\omega}; o, p, p')$ is true.

(v) Additionally, we introduce the auxiliary relation In^0 that associates instances with their In-records, that is $In^0(o; \bar{s})$ is true iff the instance o has input record $In(\bar{s})$. With slight abuse of notation, we use $\bar{\omega}$ to denote also the corresponding greedy closed execution Υ.

In the following we define a Datalog program that computes the predicates introduced above for all possible greedy executions. The program uses stratified negation.

Initialization. For each relation R in $\mathcal{P}$ we introduce the *initialization rule* $R^0(\overline{X}) \leftarrow R(\overline{X})$ to store what holds before any critical step is made. Then we add the fact rule $State^0(p_1, \ldots, p_k) \leftarrow \mathit{true}$ if in the initial configuration o_1 is at place p_1, o_2 at p_2, and so on.

Greedy Sequence: Traversal Rules. Next, we introduce rules that compute enabled graphs. The relation $Reach^i$ contains the transitive closure of the enabled graph $N_{\bar{\omega},o}$ for each o and $\bar{\omega}$, encoding a greedy execution of length i. First, a transition t from q to be p gives rise to an edge in the enabled graph $N_{\bar{\omega},o}$ if instance o can traverse that t:

$$Reach^i(\overline{W}; O, q, p) \leftarrow E_t^i(\overline{W}; O).$$

Here, $E_t^i(\overline{W}; O)$ is a shorthand for the condition obtained from E_t by replacing $In(\bar{s})$ with $In^0(O; \bar{s})$ and by replacing each atom $R(\bar{v})$ with $R^i(\overline{W}; \bar{v})$. The tuple $\overline{W}$ consists of $2i$ many distinct variables to match every critical execution with i steps. It ensures that only facts produced by $\overline{W}$ are considered. The transitive closure is computed with the following rule:

$$Reach^i(\overline{W}; O, P_1, P_3) \leftarrow Reach^i(\overline{W}; O, P_1, P_2), Reach^i(\overline{W}; O, P_2, P_3).$$

Based on $Reach^i$, SCC^i is computed by including every transition t from q to p that an instance can reach, traverse, and from where it can return to the current place:

$$SCC^i(\overline{W}; O, P, t) \leftarrow Reach^i(\overline{W}; O, P, q), E_t^i(\overline{W}; O), Reach^i(\overline{W}; O, p, P).$$

Critical Steps: Traversal Rules. We now want to record how an instance makes a critical step. An instance o_j can traverse transition t from q to p at the critical step $i+1$ if

(i) o_j is at some place in $N^q_{\bar{\omega},o_j}$ at step i,
(ii) it satisfies the execution condition E_t,
(iii) and by traversing t it leaves the current SCC.

The following *traversal rule* captures this:

$$State^{i+1}(\overline{W}, o_j, t; P_1, \ldots, P_{j-1}, p, P_{j+1}, \ldots, P_k) \leftarrow$$
$$State^i(\overline{W}; P_1, \ldots, P_{j-1}, P, P_{j+1}, \ldots, P_k), Reach^i(\overline{W}; o_j, P, q), Reach^i(\overline{W}; o_j, q, P), \quad (1)$$
$$E_t^i(\overline{W}; o_j), \neg SCC^i(\overline{W}; o_j, P, t). \quad (2)$$

Here, the condition (i) is encoded in line (1), and (ii) and (iii) are encoded in line (2).

Generation Rules. A fact in R^{i+1} may hold because

(i) it has been produced by the current greedy sequence or by the last critical step, or
(ii) by some of the previous sequences or steps.

Facts produced by previous sequences or steps are propagated with the *copy rule*: $R^{i+1}(\overline{W}, O, T; \overline{X}) \leftarrow State^{i+1}(\overline{W}, O, T; \_), R^i(\overline{W}; \overline{X})$, copying facts $R(\overline{X})$ holding after $\overline{W}$ to all extensions of $\overline{W}$.

Then we compute the facts produced by the next greedy sequence. For each instance o_j, being at some place p_j after the last critical step in $\bar{\omega}$, and for each transition t that is in $N^{p_j}_{\bar{\omega},o_j}$, with writing rule $R(\bar{u}) \leftarrow B_t(\bar{u})$, we introduce the following *greedy generation rule*:

$$R^i(\overline{W}; \bar{u}) \leftarrow State^i(\overline{W}; \_, \ldots, \_, P_j, \_, \ldots, \_), SCC^i(\overline{W}; o_j, P_j, t), B_t^i(\overline{W}; o_j; \bar{u}),$$

where condition $B_t^i(\overline{W}; O; \bar{u})$ is obtained similarly as $E_t^i(\overline{W}; O)$. In other words, all transitions t that are in $N^{p_j}_{\bar{\omega},o_j}$ are fired simultaneously, and this is done for all instances.

The facts produced at the next critical step by traversing t, which has the writing rule $R(\bar{u}) \leftarrow B_t(\bar{u})$, are generated with the *critical generation rule*: $R^{i+1}(\overline{W}, O, t; \bar{u}) \leftarrow State^{i+1}(\overline{W}, O, t; \_), B_t^i(\overline{W}; O; \bar{u})$.

Let $\Pi^{po,cl}_{\mathcal{P},\mathcal{I}}$ be the program encoding the positive closed $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ as described above.

▶ **Lemma 5.** *Let $\bar{\omega}$ be a greedy execution in the positive closed $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ of length i and $R(\bar{s})$ be a fact. Then $R(\bar{s})$ is produced by $\bar{\omega}$ iff $\Pi^{po,cl}_{\mathcal{P},\mathcal{I}} \cup \mathcal{D} \models R^i(\bar{\omega}; \bar{s})$.*

Test Program. We want to test the stability of $Q(\overline{X}) \leftarrow R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$. We collect all potential Q-answers using the relation Q'. A new query answer may be produced by an execution of any size i up to mk. Thus, for each execution of a size i from 0 to mk we introduce the Q'-rule

$$Q'(\overline{X}) \leftarrow R_1^i(\overline{W}; \bar{u}_1), \ldots, R_n^i(\overline{W}; \bar{u}_n). \tag{3}$$

Then, if there is a new query answer, the *test rule* "$\mathit{Instable} \leftarrow Q'(\overline{X}), \neg Q(\overline{X})$" fires the fact *Instable*. Let $\Pi^{test}_{\mathcal{P},\mathcal{I},Q}$ be the test program that contains Q, the Q'-rules, and the test rule.

▶ **Theorem 6.** *Q is instable in the positive closed $\mathcal{B}$ iff $\Pi^{po,cl}_{\mathcal{P},\mathcal{I}} \cup \mathcal{D} \cup \Pi^{test}_{\mathcal{P},\mathcal{I},Q} \models \mathit{Instable}$.*

Data and Process Complexity. Since $\Pi^{po,cl}_{\mathcal{P},\mathcal{I}} \cup \mathcal{D} \cup \Pi^{test}_{\mathcal{P},\mathcal{I},Q}$ is a Datalog program with stratified negation, for which reasoning is as complex as for positive Datalog, we obtain as upper bounds ExpTime for process and combined complexity, and PTime for data complexity [9]. We show that these are also lower bounds, even for singleton MDBPs. This reduction can also be adapted for acyclic fresh MDBPs, which we study in Section 8.

▶ **Lemma 7.** *Stability is* ExpTime*-hard in process and* PTime*-hard in data complexity for*

(a) *positive singleton MDBPs under closed executions, and*
(b) *positive acyclic fresh MDBPs.*

Proof Sketch.

(a) We encode query answering over a Datalog program $\Pi \cup \mathcal{D}$ into stability checking. Let A be a fact. We construct a positive singleton MDBP $\langle \mathcal{P}^{po,cl}_{\Pi,A}, \mathcal{I}_0, \mathcal{D} \rangle$, where there is a transition for each rule and the single process cycles to produce the least fixed point (LFP) of the program. In addition, the MDBP inserts the fact *dummy* if A is in the LFP. Then test query $Q_{test} \leftarrow \mathit{dummy}$ is stable in $\langle \mathcal{P}^{po,cl}_{\Pi,A}, \mathcal{I}_0, \mathcal{D} \rangle$ iff $\Pi \cup \mathcal{D} \not\models A$.
(b) Analogous, letting fresh instances play the role of the cycling singleton instance.

◀

Instance Complexity. Instance complexity turns out to be higher than data complexity. Already for acyclic positive closed MDBPs it is co-NP-hard because

(i) process instances may non-deterministically choose a transition, which creates exponentially many combinations, even in the acyclic variant; and
(ii) instances may interact by reading data written by other instances.

▶ **Lemma 8.** *There exist a positive acyclic process model $\mathcal{P}_0$, a database $\mathcal{D}_0$, and a test query Q_{test} with the following property: for every graph G one can construct an instance part $\mathcal{I}_G$ such that G is not 3-colorable iff Q_{test} is stable in $\langle \mathcal{P}_0, \mathcal{I}_G, \mathcal{D}_0 \rangle$ under closed executions.*

Clearly, Lemma 8 implies that checking stability for closed MDBPs is co-NP-hard in instance complexity. According to Theorem 11 (Section 6), instance complexity is co-NP for all closed MDBPs, which implies co-NP-completeness even for the acyclic variant.

Query Complexity. To analyze query complexity we first show how difficult it is to check whether a query returns the same answer over a database and an extension of that database.

▶ **Lemma 9** (Answer Difference)**.** *For every two fixed databases $\mathcal{D} \subseteq \mathcal{D}'$, checking whether a given conjunctive query Q satisfies $Q(\mathcal{D}) = Q(\mathcal{D}')$ is in Π^{P}_2 in the query size. Conversely, there exist databases $\mathcal{D}_0 \subseteq \mathcal{D}'_0$ such that checking for a conjunctive query Q whether $Q(\mathcal{D}_0) = Q(\mathcal{D}'_0)$ is Π^{P}_2-hard in the query size.*

Proof Idea. The first claim holds since one can check $Q(\mathcal{D}) \subsetneq Q(\mathcal{D}')$ in NP using an NP oracle. We show the second by reducing the *3-coloring extension* problem for graphs [2]. ◀

Building upon Lemma 9, we can define an MDBP that starting from $\mathcal{D}_0$ produces $\mathcal{D}'_0$. In fact, for such an MDBP it is enough to consider the simplest variants of rowo.

▶ **Proposition 10.** *Checking stability is* Π^{P}_2*-hard for*
(a) *positive fresh acyclic rowo MDBPs, and*
(b) *positive closed acyclic rowo singleton MDBPs.*

Given $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$, there are finitely many maximal extensions $\mathcal{D}'$ of $\mathcal{D}$ that can be produced by $\mathcal{B}$. We can check stability of a query Q by finitely many checks whether $Q(\mathcal{D}) = Q(\mathcal{D}')$. Since each such check is in Π^{P}_2, according to Lemma 9, the entire check is in Π^{P}_2. Thus, stability is Π^{P}_2-complete in query complexity.

6 Closed MDBPs

In the presence of negation, inserting new facts may disable transitions. During an execution, a transition may switch many times between being enabled and disabled, and greedy executions could have exponentially many critical steps. An encoding along the ideas of Section 5 would lead to a program of exponential size. This would give us an upper bound of double exponential time for combined complexity. Instead, we establish a correspondence between stability and brave query answering for Datalog with (unstratified) negation under *stable model semantics* (SMS) [9]. Due to lack of space we only state the results.

▶ **Theorem 11.** *For every closed MDBP* $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ *and every query* Q *one can construct a Datalog program with negation* $\Pi^{cl}_{\mathcal{P}}$*, based on* $\mathcal{P}$*, a database* $\mathcal{D}_{\mathcal{I}}$*, based on* $\mathcal{D}$ *and* $\mathcal{I}$*, and a test program* Π^{test}_{Q}*, based on* Q*, such that the following holds:*

$$Q \text{ is instable in } \mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle \quad \text{iff} \quad \Pi^{cl}_{\mathcal{P}} \cup \mathcal{D}_{\mathcal{I}} \cup \Pi^{test}_{Q} \models_{brave} \text{Instable}.$$

Proof Idea. For the same reason as in the positive variant, it is sufficient to consider executions of maximal length $mkrc^a$. Program $\Pi^{cl}_{\mathcal{P}}$ contains two parts:
(i) a program that generates a linear order of size $mkrc^a$ (with parameters m, k, r, c, a defined as in Section 5), starting from an exponentially smaller order, that is used to enumerate execution steps, and
(ii) a program that "guesses" an execution of size up to $mkrc^a$ by selecting for each execution step one instance and one transition, and that produces the facts that would be produced by the guessed execution. Then each execution corresponds to one stable model. The test program Π^{test}_{Q} checks if any of the guessed executions yields a new query answer. ◀

In Theorem 11, the process is encoded in the program rules while data and instances are encoded as facts. Since brave reasoning under SMS is NExpTime in program size and NP in data size [9], we have that process and combined complexity are in co-NExpTime, and data and instance complexity are in co-NP. From this and Lemma 8 it follows that instance complexity is co-NP-complete. To show that stability is co-NExpTime-complete in process and co-NP-complete in data complexity we encode brave reasoning into stability. Query complexity is Π^{P}_2-complete for the same reasons as in the positive variant.

▶ **Theorem 12.** *For every Datalog program* $\Pi \cup \mathcal{D}$*, possibly with negation, and fact* A*, one can construct a singleton MDBP* $\langle \mathcal{P}_{\Pi,A}, \mathcal{I}_0, \mathcal{D} \rangle$ *such that for the query* $Q_{test} \leftarrow$ *dummy we have:* $\Pi \cup D \models_{brave} A$ *iff* Q_{test} *is stable in* $\langle \mathcal{P}_{\Pi,A}, \mathcal{I}_0, \mathcal{D} \rangle$ *under closed executions.*

7 Acyclic Closed MDBPs

If a process net is cycle-free, all closed executions have finite length. More specifically, in an acyclic MDBP with m transitions and k running instances, the maximal length of an execution is mk. Based on this observation, we modify the encoding for the positive closed variant in Section 5 so that it can cope with negation and exploit the absence of cycles.

For an acyclic MDBP, there cannot exist any greedy steps, which would stay in a strongly connected component of the net. Therefore, we drop the encodings of greedy traversals and the greedy generation rules. We keep the rules for critical steps, but drop the atoms of relations $Reach^i$ and SCC^i. In contrast to the positive closed variant, we may have negation in the conditions E_t and B_t. However, the modified Datalog program is non-recursive, since each relation R^i and $State^i$ is defined in terms of R^j's and $State^j$'s where $j < i$.

Let $\Pi^{ac,cl}_{\mathcal{P},\mathcal{I}}$ be the program encoding an acyclic $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ as described above and let $\Pi^{test}_{\mathcal{P},\mathcal{I},Q}$ be the test program as in the cyclic variant.

▶ **Theorem 13.** *Q is instable in the closed acyclic $\mathcal{B}$ iff* $\Pi^{ac,cl}_{\mathcal{P},\mathcal{I}} \cup \mathcal{D} \cup \Pi^{test}_{\mathcal{P},\mathcal{I},Q} \models \mathit{Instable}$.

Complexity. As upper bounds for combined and data complexity, the encoding gives us the analogous bounds for non-recursive Datalog$^\neg$ programs, that is, PSPACE in combined and AC^0 in data complexity [9]. Already in the positive variant, we inherit PSPACE-hardness of process complexity (and therefore also of combined complexity) from the program complexity of non-recursive Datalog. We obtain matching lower bounds by a reverse encoding.

▶ **Lemma 14.** *For every non-recursive Datalog program Π and every fact A, one can construct a singleton acyclic positive MDBP $\langle \mathcal{P}_{\Pi,A}, \mathcal{C}_0 \rangle$ such that for the query $Q_{test} \leftarrow dummy$ we have: $\Pi \not\models A$ iff Q_{test} is stable in $\langle \mathcal{P}_{\Pi,A}, \mathcal{C}_0 \rangle$ under closed executions.*

We observe that for closed executions, the cycles increase the complexity, and moreover, cause a split between variants with and without negation. Lemma 8 and Theorem 11 together imply that instance complexity is CO-NP-complete. Query complexity is Π^P_2-complete for the same reasons as in other closed variants.

8 Positive Fresh MDBPs

All decidable variants of MDBPs that we investigated until now were so because we allowed only closed executions. In this and the next section we show that decidability can also be guaranteed if conditions and rules are positive, or if relations are divided into read and write relations (rowo). We look first at the case where initially there are no running instances.

When fresh instances start, their input can bring an arbitrary number of new constants into the database. Thus, processes can produce arbitrarily many new facts. First we show how infinitely many executions of a positive or rowo MDBP can be faithfully abstracted to finitely many over a simplified process such that a query is stable over the original process iff it is stable over the simplified one. For such simplified positive MDBPs, we show how to encode stability checking into query answering in Datalog.

Abstraction Principle. Let $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D}, \tau_{\mathcal{B}} \rangle$ be a positive or rowo MDBP and let Q be a query that we want to check for stability. Based on $\mathcal{B}$ and Q we construct an MDBP $\mathcal{B}' = \langle \mathcal{P}', \mathcal{I}, \mathcal{D}, \tau_{\mathcal{B}} \rangle$ that has the same impact on the stability of Q but uses at most linearly many fresh values from the domain.

Let *adom* be the active domain of $\mathcal{B}$ and Q, that is the set of all constants appearing in $\mathcal{B}$ and Q. Let $\tau_1, \ldots, \tau_n$ be all timestamps including $\tau_{\mathcal{B}}$ that appear in comparisons in $\mathcal{B}$ such that $\tau_i < \tau_{i+1}$. We introduce $n+1$ many fresh timestamps $\tau'_0, \ldots, \tau'_n \notin adom$ such that $\tau'_0 < \tau_1 < \tau'_1 < \cdots < \tau_n < \tau'_n$. If there are no comparisons in $\mathcal{B}$ we introduce one fresh timestamp τ'_0. Further, let a be a fresh value such that $a \notin adom$. Let $adom^* = adom \cup \{\tau'_0, \ldots, \tau'_n\} \cup \{a\}$ be the *extended active domain*.

Then, we introduce the *discretization* function $\delta_{\mathcal{B}} \colon dom_{\mathbb{Q}^+} \to dom_{\mathbb{Q}^+}$ that based on $adom^*$ "discretizes" $dom_{\mathbb{Q}^+}$ as follows: for each $\tau \in \mathbb{Q}^+$

(i) $\delta_{\mathcal{B}}(\tau) = \tau$ if $\tau = \tau_i$ for some i;
(ii) $\delta_{\mathcal{B}}(\tau) = \tau'_i$ if $\tau_i < \tau < \tau_{i+1}$ for some i;
(iii) $\delta_{\mathcal{B}}(\tau) = \tau'_0$ if $\tau < \tau_1$;
(iv) and $\delta_{\mathcal{B}}(\tau) = \tau'_n$ if $\tau_n < \tau$;
(v) and for $c \in dom$ if $c \in adom^*$ then $\delta_{\mathcal{B}}(c) = c$; otherwise $\delta_{\mathcal{B}}(c) = a$.

If $\mathcal{B}$ has no comparisons then $\delta_{\mathcal{B}}(\tau) = \tau'_0$ for each τ. We extend $\delta_{\mathcal{B}}$ to all syntactic objects containing constants, including executions. Now, we define $\mathcal{P}'$ to be as $\mathcal{P}$, except that we add conditions on each outing transition from *start* such that only instances with values from $adom^*$ can traverse, and instances with the timestamps greater or equal than $\tau_{\mathcal{B}}$.

▶ **Proposition 15** (Abstraction). *Let $\Upsilon = \mathcal{C} \rightsquigarrow \mathcal{C}_1 \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_m$ be an execution in $\mathcal{B}$ that produces a set of facts W, and let $\Upsilon' = \delta_{\mathcal{B}}\Upsilon = \delta_{\mathcal{B}}\mathcal{C} \rightsquigarrow \delta_{\mathcal{B}}\mathcal{C}_1 \rightsquigarrow \cdots \rightsquigarrow \delta_{\mathcal{B}}\mathcal{C}_m$. Further, let Υ'' be an execution in $\mathcal{B}'$. Then the following holds:*

(a) *Υ' is an execution in $\mathcal{B}'$ that produces $\delta_{\mathcal{B}}W$;*
(b) *$Q(\mathcal{D}) \neq Q(\mathcal{D} \cup W)$ iff $Q(\mathcal{D}) \neq Q(\mathcal{D} \cup \delta_{\mathcal{B}}W)$;*
(c) *Υ'' is an execution in $\mathcal{B}$.*

In other words, each execution in $\mathcal{B}$ can be $\delta_{\mathcal{B}}$-abstracted and it will be an execution in $\mathcal{B}'$, and more importantly, an execution in $\mathcal{B}$ produces a new query answer if and only if the $\delta_{\mathcal{B}}$-abstracted version produces a new query answer in $\mathcal{B}'$.

Encoding into Datalog. Since $\mathcal{B}'$ allows only finitely many new values in fresh instances, there is a bound on the maximal extensions of $\mathcal{D}$ that can be produced. Moreover, since there is no bound on the number of fresh instances that can start, there is only a single maximal extension of $\mathcal{D}$, say $\mathcal{D}'$, that can result from $\mathcal{B}'$. We now define the program $\Pi^{po,fr}_{\mathcal{P},Q} \cup \mathcal{D}$ whose least fixpoint is exactly this $\mathcal{D}'$.

First, we introduce the relations that we use in the encoding. To record which fresh instances can reach a place p in $\mathcal{P}$, we introduce for each p a relation In_p with the same arity as In. That is, $In_p(\bar{s})$ evaluates to true in the program iff an instance with the input record $In(\bar{s})$ can reach p. As in the closed variant, we use a primed version R' for each relation R to store R-facts produced by the process.

Now we define the rules. Initially, all relevant fresh instances (those with constants from $adom^*$) sit at the *start* place. We encode this by the *introduction rule*: $In_{start}(X_1, \ldots, X_n) \leftarrow adom^*(X_1), \ldots, adom^*(X_n)$. Here, with slight abuse of notation, $adom^*$ represents a unary relation that we initially instantiate with the constants from $adom^*$. Also initially, we make a primed copy of each database fact, that is, for each relation R in $\mathcal{P}$ we define the *copy rule*: $R'(\overline{X}) \leftarrow R(\overline{X})$.

Then we encode instance traversals. For every transition t that goes from a place q to a place p, we introduce a *traversal rule* that mimics how instances having reached q move on to p, provided their input record satisfies the execution condition for t. Let $E_t = In(\bar{s}), R_1(\bar{s}_1), \ldots, R_l(\bar{s}_l), G_t$ be the execution condition for t, where G_t comprises the comparisons. We

define the condition $E'_t(\bar{s})$ as $In_q(\bar{s}), R'_1(\bar{s}_1), \ldots, R'_l(\bar{s}_l), G_t$, obtained from E_t by renaming the *In*-atom and priming all database relations. Then, the *traversal rule* for t is: $In_p(\bar{s}) \leftarrow E'_t(\bar{s})$. Here, $E'_t(\bar{s})$ is defined over the primed signature since a disabled transition may become enabled as new facts are produced.

Which facts are produced by traversing t is captured by a *generation rule*. Let W_t: $R(\bar{u}) \leftarrow B_t(\bar{u})$ be the writing rule for t, with the query $B_t(\bar{u}) \leftarrow In(\bar{s}'), R_1(\bar{s}'_1), \ldots, R_n(\bar{s}'_n), M_t$, where M_t comprises the comparisons. Define $B'_t(\bar{s}', \bar{u}) \leftarrow In_q(\bar{s}'), R'_1(\bar{s}'_1), \ldots, R'_n(\bar{s}'_n), M_t$. The corresponding generation rule is $R'(\bar{u}) \leftarrow E'_t(\bar{s}), B'_t(\bar{s}', \bar{u}), \bar{s} = \bar{s}'$, which combines the constraints on the instance record from E_t and W_t.

Let $\Pi^{po,fr}_{\mathcal{P},Q}$ be the program defined above, encoding the positive fresh $\mathcal{B}'$ obtained from $\mathcal{B}$. The program is constructed in such a way that it computes exactly the atoms that are in the maximal extension $\mathcal{D}'$ of $\mathcal{D}$ produced by $\mathcal{B}'$. Let $R'(\bar{v})$ be a fact.

▶ **Lemma 16.** *There is an execution in the positive fresh $\mathcal{B}$ producing $R(\bar{v})$ iff $\Pi^{po,fr}_{\mathcal{P},Q} \cup \mathcal{D} \models R'(\bar{v})$.*

Let Π^{test}_{Q} be defined like $\Pi^{test}_{\mathcal{P},\mathcal{I},Q}$ in Section 5, except that there is only one rule for Q', obtained from (3) by replacing R^i_j with R'_j. Then Proposition 15 and Lemma 16 imply:

▶ **Theorem 17.** *Q is instable the positive fresh $\mathcal{B}$ iff $\Pi^{po,fr}_{\mathcal{P},Q} \cup \mathcal{D} \cup \Pi^{test}_{Q} \models Instable$.*

Complexity. Since $\Pi^{po,fr}_{\mathcal{P},Q} \cup \mathcal{D} \cup \Pi^{test}_{Q}$ is a program with stratified negation, stability checking over positive fresh MDBPs is in ExpTime for process and combined complexity, and in PTime for data complexity [9]. From Lemma 7 we know that these are also lower bounds for the respective complexity measures. Query complexity is Π^{P}_2-complete as usual, and instance complexity is trivial for fresh processes.

Positive MDBPs. To reason about arbitrary positive MDBPs, we can combine the encoding for the fresh variant ($\Pi^{po,fr}_{\mathcal{P},Q}$) from this section and the one for the closed variant from Section 5 ($\Pi^{po,cl}_{\mathcal{P},\mathcal{I}}$). The main idea is that to obtain maximal extensions, each greedy execution sequence is augmented by also flooding the process with fresh instances. The complexities for the full positive variant are inherited from the closed variant.

9 Read-Only-Write-Only MDBPs

In general MDBPs, processes can perform recursive inferences by writing into relations from which they have read. It turns out that if relations are divided into read-only and write-only, the complexity of stability reasoning drops significantly.

The main simplifications in this case are that

(i) one traversal per instance and transition suffices, since no additional fact can be produced by a second traversal;

(ii) instead of analyzing entire executions, it is enough to record which paths an individual process instance can take and which facts it produces, since instances cannot influence each other.

As a consequence, the encoding program can be non-recursive and it is independent of the instances in the process configuration. A complication arises, however, since the maximal extensions of the original database $\mathcal{D}$ by the MDBP $\mathcal{B}$ are not explicitly represented by this approach. They consist of unions of maximal extensions by each instance and are encoded into the test query, which is part of the program.

▶ **Theorem 18.** *For every rowo MDBP $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ and query Q one can construct a nonrecursive Datalog program $\Pi^{ro}_{\mathcal{P},Q}$, based on $\mathcal{P}$ and Q, and a database instance $\mathcal{D}_{\mathcal{I}}$, based on $\mathcal{D}$ and $\mathcal{I}$, such that: Q is instable in $\mathcal{B}$ iff $\Pi^{ro}_{\mathcal{P},Q} \cup \mathcal{D}_{\mathcal{I}} \models$ Instable.*

From the theorem it follows that data and instance complexities are in AC^0, except for instance complexity in fresh variants, for which it is constant.

Process, Query and Combined Complexity. Since CQ evaluation can be encoded into an execution condition, this gives us co-NP-hardness of stability in process complexity. We also show that it is in co-NP. First we note that due to the absence of recursion, one can check in NP whether a set of atoms is produced by a process instance.

▶ **Proposition 19.** *Let $\mathcal{B}$ be a singleton rowo MDBP. One can decide in* NP*, whether for given facts $A_1, \ldots, A_m$, there is an execution in $\mathcal{B}$ that produces $A_1, \ldots, A_m$.*

Next, suppose that $\mathcal{I}$, $\mathcal{D}$ and $Q(\bar{v}) \leftarrow B_1, \ldots, B_m$ are a fixed instance part, database and query. Given a process model $\mathcal{P}$, we want to check that Q is instable in $\mathcal{B}_{\mathcal{P}} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$. Making use of the abstraction principle for fresh constants, we can guess in polynomial time an instantiation $B'_1, \ldots, B'_n$ of the body of Q that returns an answer not in $Q(\mathcal{D})$. Then we verify that $B'_1, \ldots, B'_n$ are produced by $\mathcal{B}_{\mathcal{P}}$. Such a verification is possible in NP according to Proposition 19. We guess a partition of the set of facts $B'_1, \ldots, B'_n$, guess one instance, possibly fresh, for each component set of the partition, and verify that the component set is produced by the instance. Since all verification steps were in NP, the whole check is in NP.

Query complexity is Π^{P}_2-complete for the same reasons as in the general variant, and one can show that this is also the upper-bound for the combined complexity.

10 Related Work

Traditional approaches for business process modeling focus on the set of activities to be performed and the flow of their execution. These approaches are known as *activity-centric*. A different perspective, mainly investigated in the context of databases, consists in identifying the set of data (entities) to be represented and describes processes in terms of their possible evolutions. These approaches are known as *data-centric*.

In the context of activity-centric processes, Petri Nets (PNs) have been used for the representation, validation and verification of formal properties, such as absence of deadlock, boundedness and reachability [26, 27]. In PNs and their variants, a token carries a limited amount of information, which can be represented by associating to the token a set of variables, like in colored PNs [18]. No database is considered in PNs.

Among data-centric approaches, *Transducers* [1, 25] were among the first formalisms ascribing a central role to the data and how they are manipulated. These have been extended to *data driven web systems* [11] to model the interaction of a user with a web site, which are then extended in [10]. These frameworks express insertion and deletion rules using FO formulas. The authors verify properties expressed as FO variants of LTL, CTL and CTL* temporal formulas. The verification of these formulas results to be undecidable in the general case. Decidability is obtained under certain restrictions on the input, yielding to ExpSpace complexity for checking LTL formulas and co-NExpTime and ExpSpace for CTL and CTL* resp., in the propositional case.

Data-Centric Dynamic Systems (DCDSs) [4] describe processes in terms of guarded FO rules that evolve the database. The authors study the verification of temporal properties

expressed in variants of μ-calculus (that subsumes CTL*-FO). They identify several undecidable classes and isolate decidable variants by assuming a bound on the size of the database at each step or a bound on the number of constants at each run. In these cases verification is ExpTime-complete in data complexity.

Overall, both frameworks are more general than MDBPs, since deletions and updates of facts are also allowed. This is done by rebuilding the database after each execution step. Further, our stability problem can be encoded as FO-CTL formula. However, our decidability results for positive MDBPs are not captured by the decidable fragments of those approaches. In addition, the authors of the work above investigate the borders of decidability, while we focus on a simpler problem and study the sources of complexity. Concerning the process representation, both approaches adopt a rule-based specification. This makes the control flow more difficult to grasp, in contrast to activity-centric approaches where the control flow has an explicit representation.

Artifact-centric approaches [17] use artifacts to model business relevant entities. In [6, 14, 15] the authors investigate the verification of properties of artifact-based processes such as reachability, temporal constraints, and the existence of dead-end paths. However, none of these approaches explicitly models an underlying database. Also, the authors focus on finding suitable restrictions to achieve decidability, without a fine-grained complexity analysis as in our case.

Approaches in [3] and [5], investigate the challenge of combining processes and data, however, focusing on the problem of data provenance and of querying the process structure.

In [12, 20] the authors study the problem of determining if a query over views is independent from a set of updates over the database. The authors do not consider a database instance nor a process. Decidability in rowo MDBPs can be seen as a special case of those.

In summary, our approach to process modeling is closer to the activity-centric one but we model manipulation of data like in the data-centric approaches. Also, having process instances and MDBPs restrictions gives finer granularity compared to data-centric approaches.

11 Discussion and Conclusion

Discussion. An interesting question is how complex stability becomes if MDBPs are not monotonic, i.e., if updates or deletions are allowed. In particular, for positive MDBPs we can show the following. In acyclic positive closed MDBPs updates and deletions can be modeled using negation in the rules, thus stability stays PSpace-complete. For the cyclic positive closed variant, allowing updates or deletions is more powerful than allowing negation, and stability jumps to ExpSpace-completeness. For positive MDBPs with updates or deletions stability is undecidable.

In case the initial database is not known, our techniques can be still applied since an arbitrary database can be produced by fresh instances starting from an empty database.

Contributions. Reasoning about data and processes can be relevant in decision support to understand how processes affect query answers.

1. To model processes that manipulate data we adopt an explicit representation of the control flow as in standard BP languages (e.g., BPMN). We specify how data is manipulated as annotations on top of the control flow.
2. Our reasoning on stability can be offered as a reasoning service on top of the query answering that reports on the reliability of an answer. Ideally, reasoning on stability should not bring a significant overhead on query answering in practical scenarios. Existing

work on processes and data [4] shows that verification of general temporal properties is typically intractable already measured in the size of the data.

3. In order to identify tractable cases and sources of complexity we investigated different variants of our problem, by considering negation in conditions, cyclic executions, read access to written data, presence of pending process instances, and the possibility to start fresh process instances.
4. Our aim is to deploy reasoning on stability to existing query answering platforms such as SQL and ASP [19]. For this reason we established different encodings into suitable variants of Datalog, that are needed to capture the different characteristics of the problem. For each of them we showed that our encoding is optimal. In contrast to existing approaches, which rely on model checking to verify properties, in our work we rely on established database query languages.

Open Questions. In our present framework we cannot yet model process instances with activities that are running in parallel. Currently, we are able to deal with it only in case instances do not interact (like in rowo). Also, we do not know yet how to reason about expressive queries, such as conjunctive queries with negated atoms, and first-order queries. From an application point of view, stability of aggregate queries and aggregates in the process rules are relevant. A further question is how to quantify instability, that is, in case a query is not stable, how to compute the minimal/maximal number of possible new answers.

Acknowledgments. This work was partially supported by the research projects MAGIC, funded by the province of Bozen-Bolzano, and CANDy and PARCIS, funded by the Free University of Bozen-Bolzano.

References

1 S. Abiteboul, V. Vianu, B.S. Fordham, and Y. Yesha. Relational Transducers for Electronic Commerce. In *PODS*, pages 179–187, 1998. doi:10.1145/275487.275507.

2 M. Ajtai, R. Fagin, and L.J. Stockmeyer. The Closure of Monadic NP. *J. Comput. Syst. Sci.*, 60(3):660–716, 2000.

3 Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting Lipstick on Pig: Enabling Database-Style Workflow Provenance. *PVLDB*, 5(4):346–357, 2011.

4 B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of Relational Data-Centric Dynamic Systems with External Services. In *PODS*, pages 163–174, 2013. doi:10.1145/2463664.2465221.

5 C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying Business Processes. In *VLDB*, pages 343–354, 2006.

6 K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In *BPM*, pages 288–304, 2007.

7 Bonitasoft. Bonita BPM. Accessed: 2015-12-16. URL: http://www.bonitasoft.com.

8 G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving Data Quality: Consistency and Accuracy. In *VLDB*, pages 315–326, 2007.

9 E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

10 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic Verification of Data-Centric Business Processes. In *ICDT*, pages 252–267, 2009. doi:10.1145/1514894.1514924.

11 A. Deutsch, L. Sui, and V. Vianu. Specification and Verification of Data-Driven Web Services. In *PODS*, pages 71–82, 2004.

12 C. Elkan. Independence of Logic Database Queries and Updates. In *PODS*, pages 154–160, 1990. doi:10.1145/298514.298557.

13 W. Fan, F. Geerts, and J. Wijsen. Determining the Currency of Data. *ACM Trans. Database Syst.*, 37(4):25, 2012.

14 C. E. Gerede, K. Bhattacharya, and J. Su. Static Analysis of Business Artifact-Centric Operational Models. In *SOCA*, pages 133–140, 2007.

15 C. E. Gerede and J. Jianwen Su. Specification and Verification of Artifact Behaviors in Business Process Models. In *ICSOC*, pages 181–192, 2007.

16 F. T. Heath, D. Boaz, M. Gupta, R. Vaculín, Y. Sun, R. Hull, and L. Limonad. Barcelona: A Design and Runtime Environment for Declarative Artifact-Centric BPM. In *ICSOC*, pages 705–709, 2013. doi:10.1007/978-3-642-45005-1_65.

17 R. Hull. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In *OTM*, pages 1152–1163, 2008.

18 K. Jensen and L.M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.

19 N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

20 A.Y. Levy and Y. Sagiv. Queries Independent of Updates. In *VLDB*, pages 171–181, 1993.

21 E. Marengo, W. Nutt, and O. Savković. Towards a Theory of Query Stability in Business Processes. In *AMW*, volume 1189 of *CEUR Workshop Proceedings*, 2014.

22 Object Management Group. *Business Process Model and Notation 2.0 (BPMN)*, Jan 2011. URL: http://www.omg.org/spec/BPMN/2.0/.

23 S. Razniewski and W. Nutt. Completeness of Queries over Incomplete Databases. *PVLDB*, 4(11):749–760, 2011.

24 O. Savković, E. Marengo, and W. Nutt. Query Stability in Data-aware Business Processes. Technical Report KRDB15-1, KRDB Research Center, Free Univ. Bozen-Bolzano, 2015. URL: http://www.inf.unibz.it/krdb/pub/tech-rep.php.

25 M. Spielmann. Verification of Relational Transducers for Electronic Commerce. In *PODS*, pages 92–103. ACM, 2000.

26 W.M.P. van der Aalst. Verification of Workflow Nets. In *ICATPN*, pages 407–426, 1997. doi:10.1007/3-540-63139-9_48.

27 W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

Document Spanners: From Expressive Power to Decision Problems

Dominik D. Freydenberger[*1] **and Mario Holldack**[2]

1 **University of Bayreuth, Bayreuth, Germany**
2 **Goethe University, Frankfurt am Main, Germany**

Abstract

We examine *document spanners*, a formal framework for information extraction that was introduced by Fagin et al. (PODS 2013). A document spanner is a function that maps an input string to a relation over *spans* (intervals of positions of the string). We focus on document spanners that are defined by *regex formulas*, which are basically regular expressions that map matched subexpressions to corresponding spans, and on *core spanners*, which extend the former by standard algebraic operators and string equality selection.

First, we compare the expressive power of core spanners to three models – namely, *patterns*, *word equations*, and a rich and natural subclass of *extended regular expressions* (regular expressions with a repetition operator). These results are then used to analyze the complexity of query evaluation and various aspects of static analysis of core spanners. Finally, we examine the relative succinctness of different kinds of representations of core spanners and relate this to the simplification of core spanners that are extended with difference operators.

1998 ACM Subject Classification H.2.1 Data models, H.2.4 Textual databases, Relational databases, Rule-based databases, F.4.3 Classes defined by grammars or automata, Decision problems, F.1.1 Relations between models

Keywords and phrases Information extraction, document spanners, regular expressions, regex, patterns, word equations, decision problems, descriptional complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.17

1 Introduction

Information Extraction (IE) is the task of automatically extracting structured information from texts. This paper examines *document spanners*, a formalization of the IE query language AQL, which is used in IBM's SystemT. Document spanners were introduced by Fagin et al. [7] in order to allow the theoretical examination of AQL, and were also used in [6].

A *span* is an interval on positions of a string w, and a *spanner* is a function that maps w to a relation over spans of w. A central topic of [7] and of the present paper are *core spanners*. The primitive building blocks of core spanners are *regex formulas*, which are regular expressions with variables. Each of these variables corresponds to a subexpression, and whenever a regex formula α matches a string w, each variable is mapped to the span in w that matches that subexpression. Hence, each match of α on w determines a tuple of spans; and as there can be multiple matches of a regex formula to a string, this process creates a relation over spans of w. Core spanners are then defined by extending regex formulas with the relational operations projection, union, natural join, and string equality selection.

* Supported by Deutsche Forschungsgemeinschaft (DFG) under grant FR 3551/1-1.

One of the two main topics of the present paper is the examination of decision problems for core spanners, in particular evaluation and static analysis. These results are mostly derived from the other main topic, the examination of the expressive power of core spanners in relation to three other models that use repetition operators, which act similar to the spanners' string equality selection.

The first of these models are *patterns*. A pattern is word that consists of variables and terminals, and generates the language of all words that can be obtained by substitution of the variables with arbitrary terminal words. For example, the pattern $\alpha = xx\mathtt{ab}y$ (where x and y are variables, and $\mathtt{a}$ and $\mathtt{b}$ are terminals) generates the language of all words that have a prefix that consists of a square, followed by the word $\mathtt{ab}$. Although pattern languages have a simple definition, various decision problems for them are surprisingly hard. For example, their membership problem is NP-complete (cf. Jiang et al. [19]), and their inclusion problem is undecidable (cf. Bremer and Freydenberger [3]). As we show that core spanners can recognize pattern languages, this allows us to conclude that evaluation of core spanners is NP-hard, and that spanner containment is undecidable.

The second model we consider are *word equations*, which are equations of the form $\alpha = \beta$, where α and β are patterns, which can be used to define word relations. We show that word equations with regular constraints can express all relations that are expressible with core spanners. By using an improved version of Makanin's algorithm (cf. Diekert [5]), this allows us to show that satisfiability and hierarchicality for core spanners can be decided in PSPACE. Moreover, using coding techniques from word equations, we show that two common relations from combinatorics on words can be selected with core spanners.

The third model are *regexes* (also called *extended regular expressions* in literature). These are regular expressions that can use a repetition operator, that is available in most modern implementations for regular expressions (see, e. g., Friedl [14]) and that allows the definition of non-regular languages. For example, the regex $x\{\Sigma^*\}\&x\&x$ generates all words www with $w \in \Sigma^*$, as $x\{\Sigma^*\}$ generates some word w which is stored in the variable x, and each occurrence of $\&x$ repeats that w. As a consequence of this increase in expressive power, many decision problems are harder for regexes than for their "classical" counterparts. In particular, various problems of static analysis are undecidable (Freydenberger [11]).

But as shown by Fagin et al. [7], document spanners cannot define all languages that are definable by regexes. Intuitively, the reason for this is that regexes can use their repetition operators inside a Kleene star, which allows them to repeat an arbitrary word an unbounded number of times, while core spanners have to express repetitions with variables and string equality selections. Inspired by this observation, we introduce *variable-star free (or vstar-free) regexes* as those regexes that neither define nor use variables inside a Kleene star. We show that every vstar-free regex can be converted into an equivalent core spanner. Since all undecidability results by Freydenberger [11] also apply to vstar-free regexes, these undecidability results carry over to core spanners. This also has various consequences to the minimization and the relative succinctness of classes of spanner representations, and to the simplification of core spanners with difference operators. As a further contribution, we also develop tools to prove inexpressibility for vstar-free regular expressions and for core spanners.

As we shall see, many of the observed lower bounds hold even for comparatively restricted classes of core spanners (in particular, most of the results hold for spanners that do not use join). Hence, the authors consider it reasonable to expect that these results can be easily adapted to other information extraction languages that combine regular expressions with capture variables and a string equality operator.

In addition to regex formulas, Fagin et al. [7] also consider two types of automata as basic building blocks of spanner representations. While the present paper does not discuss these

in detail, most of the results on spanner representations that are based on regex formulas can be directly converted to the respective class of spanner representations that are based on automata.

Related work. For an overview of related models, we refer to Fagin et al. [7]. In addition to this, we highlight connections to models with similar properties. In [7], Fagin et al. showed that there is a language that can be defined by regexes, but not by core spanners. Furthermore, they compared the expressive power of core spanners and a variant of conjunctive regular path queries (CRPQs), a graph querying language. Barceló et al. [1] introduced extended CRPQs (ECRPQs), which can compare paths in the graph with regular relations. While there is no direct connection between ECRPQs and core spanners, both models share the basic idea of combining regular languages with a comparison operator that can express string equality. As shown by Freydenberger and Schweikardt [13], ECRPQs have undecidability results that are comparable to those in the present paper, and to those for regexes (cf. Freydenberger [11]). Furthermore, Barceló and Muñoz [2] have used word equations with regular constraints for variants of CRPQs.

Structure of the paper. In Section 2, we give definitions of regexes and of core spanners. Section 3 compares the expressive power of core spanners to patterns, word equations, and vstar-free regular expressions. The results from this section are then used in Section 4 to examine the complexity of evaluation and static analysis of spanners. We also examine the consequences of these results to the relative succinctness of different spanner representations. Section 5 concludes the paper. Due to space reasons, all proofs were moved to an appendix that is contained in the full version of the paper.

2 Preliminaries

Let $\mathbb{N}$ and $\mathbb{N}_{>0}$ be the sets of non-negative and positive integers, respectively. Let Σ be a fixed finite alphabet of *(terminal) symbols*. Except when stated otherwise, we assume $|\Sigma| \geq 2$. We use ε to denote the *empty word*. For every word $w \in \Sigma^*$ and every $a \in \Sigma$, let $|w|$ denote the length of w, and $|w|_a$ the number of occurrences of a in w. A word $x \in \Sigma^*$ is a *subword* of a word $y \in \Sigma^*$ if there exist $u, v \in \Sigma^*$ with $y = uxv$. A word $x \in \Sigma^*$ is a *prefix* of a word $y \in \Sigma^*$ if there exists a $v \in \Sigma^*$ with $y = xv$, and a *proper prefix* if it is a prefix and $x \neq y$. For every $n \in \mathbb{N}$, an *n-ary word relation (over Σ)* is a subset of $(\Sigma^*)^n$.

2.1 Regexes (Extended Regular Expressions)

This section introduces the syntax and semantics of regexes, which we shall also use for spanners in Section 2.2. We begin with the syntax, which follows the definition from [7].

▶ **Definition 1.** We fix an infinite set X of *variables* and define the set M of *meta symbols* as $M := \{\varepsilon, \emptyset, (,), \{, \}, \cdot, \vee, ^*, \&\}$. Let Σ, X, and M be pairwise disjoint. The set of *regexes (extended regular expressions)* is defined as follows:

1. The symbols $\emptyset$, ε, and every $a \in \Sigma$ are regexes.
2. If α_1 and α_2 are regex, then $(\alpha_1 \cdot \alpha_2)$ *(concatenation)*, $(\alpha_1 \vee \alpha_2)$ *(disjunction)*, and (α_1^*) *(Kleene star)* are regexes.
3. For every $x \in X$ and every regex α that contains neither $x\{\cdots\}$ nor $\&x$ as a subword, $x\{\alpha\}$ is a regex *(variable binding)*.
4. For every $x \in X$, $\&x$ is a regex *(variable reference)*.

If a subword β of a regex α is a regex itself, we call β a *subexpression (of α)*. The set of all subexpressions of α is denoted by $\mathsf{Sub}(\alpha)$, and the set of variables occurring in variable bindings in a regex α is denoted by $\mathsf{Vars}(\alpha)$. If a regex α contains neither variable references, nor variable bindings, we call α a *proper regular expression*.

In other words, we use the term "proper" to distinguish those expressions that are usually just called "regular expressions" from the more general extended regular expressions. We use the notation α^+ as a shorthand for $\alpha \cdot \alpha^*$. Parentheses can be added freely. We may also omit parentheses and the concatenation operator, where we assume $*$ and $+$ are taking precedence over concatenation, and concatenation precedes disjunction. Furthermore, we use Σ as a shorthand for the regular expression $\bigvee_{a \in \Sigma} a$.

Before introducing the semantics of regexes formally, we give an intuitive explanation. An expression of the form $\alpha = x\{\beta\}$ matches the same strings as β, but α additionally stores the matched string in the variable x. Using a variable reference $\&x$, this string can then be repeated. For example, let $\alpha := (x\{\Sigma^*\} \cdot \&x)$. The subexpression $x\{\Sigma^*\}$ matches any string $w \in \Sigma^*$ and stores this match in x. The following variable reference $\&x$ repeats the stored w. Thus, α defines the (non-regular) *copy-language* $\{ww \mid w \in \Sigma^*\}$.

The following definition of the semantics of regexes is based on the semantics by Freydenberger [11], which is an adaption of the semantics from Câmpeanu et al. [4] (the former uses variables, the latter backreferences). In comparison to [11], the case for Kleene star has been changed, in order to make the definition compatible with the parse trees from Fagin et al. [7].

▶ **Definition 2.** Let γ be a regex over Σ and X. A *γ-parse tree* is a finite, directed, and ordered tree T_γ. Its nodes are labeled with tuples of the form $(w, \gamma') \in (\Sigma^* \times \mathsf{Sub}(\gamma))$. The root of every γ-parse tree T_γ is labeled with (w, γ), $w \in \Sigma^*$; and the following rules must hold for each node v of T_γ:

1. If v is labeled (w, a) with $a \in (\Sigma \cup \{\varepsilon\})$, then v is a leaf, and $w = a$.
2. If v is labeled $(w, (\beta_1 \cdot \beta_2))$, then v has exactly one left child v_1 and exactly one right child v_2 with respective labels (w_1, β_1) and (w_2, β_2), and $w = w_1 w_2$.
3. If v is labeled $(w, (\beta_1 \vee \beta_2))$, then v has a single child, labeled (w, β_1) or (w, β_2).
4. If v is labeled (w, β^*), then one of the following cases holds:
   - (a) $w = \varepsilon$, and v is a leaf, or
   - (b) $w = w_1 w_2 \dots w_k$ for words $w_1, \dots, w_k \in \Sigma^+$ (with $k \geq 1$), and v has k children $v_1, \dots, v_k$ (ordered from left to right) that are labeled $(w_1, \beta), \dots, (w_k, \beta)$.
3. If v is labeled $(w, x\{\beta\})$, then v has a single child, labeled (w, β).
4. If v is labeled $(w, \&x)$, let $\prec$ denote the post-order of the nodes of T_γ (that results from a left-to-right, depth-first traversal). Then one of the following cases applies:
   - (a) If there is no node v' with $v' \prec v$ that is labeled $(w', x\{\beta'\}) \in \Sigma^* \times \mathsf{Sub}(\gamma)$, then v is a leaf, and $w = \varepsilon$.
   - (b) Otherwise, let v' be the node with $v' \prec v$ that is $\prec$-maximal among nodes labeled $(w', x\{\beta'\})$. Then v is a leaf, and $w = w'$.

If the root of a γ-parse tree T_γ is labeled (w, γ), we call T_γ a *γ-parse tree for w*. If the context is clear, we omit γ and call T_γ a parse tree.

There is no parse tree for $\emptyset$, and references to unbound variables (i. e., variables that were not assigned a value with a variable binding operator) default to ε. For an example of a parse tree, see Figure 1.

We use parse trees to define the semantics of regexes:

▶ **Definition 3.** A regex γ recognizes the language $\mathcal{L}(\gamma)$ of all $w \in \Sigma^*$ for which there exists a γ-parse tree T_γ with (w, γ) as root label.

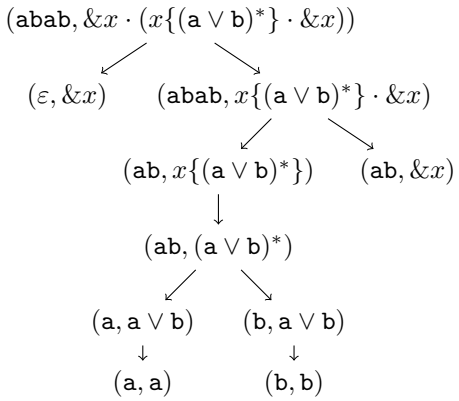


Figure 1 The α-parse tree for w, where $\alpha := \&x \cdot (x\{(\mathtt{a} \vee \mathtt{b})^*\} \cdot \&x)$ and $w := \mathtt{abab}$.

▶ **Example 4.** Let $\alpha := x\{\Sigma^+\} \cdot (\&x)^+$. Then $\mathcal{L}(\alpha) = \{w^n \mid w \in \Sigma^+, n \geq 2\}$. Furthermore, let $\beta := x\{\Sigma^+\} \cdot \&x \cdot x\{\Sigma^+\} \cdot \&x$. Then $\mathcal{L}(\beta) = \{x_1x_1x_2x_2 \mid x_1, x_2 \in \Sigma^+\}$. Finally, for some $a \in \Sigma$, let $\gamma := x\{aa^+\} \cdot (\&x)^+$. Then $\mathcal{L}(\gamma) = \{a^n \mid n \geq 2, n \text{ is not prime}\}$.

2.2 Document Spanners

Let $w := a_1a_2 \cdots a_n$ be a word over Σ, with $n \in \mathbb{N}$ and $a_1, \ldots, a_n \in \Sigma$. A *span of* w is an interval $[i, j\rangle$ with $1 \leq i \leq j \leq n + 1$ and $i, j \in \mathbb{N}$. For each span $[i, j\rangle$ of w, we define a subword $w_{[i,j\rangle} := a_i \cdots a_{j-1}$. In other words, each span describes a subword of w by its bounding indices. Two spans $[i, j\rangle$ and $[i', j'\rangle$ of w are equal if and only if $i = i'$ and $j = j'$. These spans *overlap* if $i \leq i' < j$ or $i' \leq i < j'$, and are *disjoint*, otherwise. The span $[i, j\rangle$ *contains* the span $[i', j'\rangle$ if $i \leq i' \leq j' \leq j$. The *set of all spans of* w is denoted by $\mathsf{Spans}(w)$.

▶ **Example 5.** Let $w := \mathtt{aabbcabaa}$. As $|w| = 9$, both $[3, 3\rangle$ and $[5, 5\rangle$ are spans of w, but $[10, 11\rangle$ is not. As $3 \neq 5$, the first two spans are not equal, even though $w_{[3,3\rangle} = w_{[5,5\rangle} = \varepsilon$. The whole word w is described by the span $[1, 10\rangle$.

▶ **Definition 6.** Let SVars be a fixed, infinite set of *span variables*, where Σ and SVars are disjoint. Let $V \subset \mathsf{SVars}$ be a finite subset of SVars, and let $w \in \Sigma^*$. A (V, w)-*tuple* is a function $\mu \colon V \to \mathsf{Spans}(w)$, that maps each variable in V to a span of w. If context allows, we write w-tuple instead of (V, w)-tuple. A set of (V, w)-tuples is called a (V, w)-*relation*.

As V and $\mathsf{Spans}(w)$ are finite, every (V, w)-relation is finite by definition. Our next step is the definition of spanners, which map words w to (V, w)-relations:

▶ **Definition 7.** Let V and Σ be alphabets of variables and symbols, respectively. A *(document) spanner* is a function P that maps every word $w \in \Sigma^*$ to a (V, w)-relation $P(w)$. Let V be denoted by $\mathsf{SVars}(P)$. A spanner P is n-*ary* if $|\mathsf{SVars}(P)| = n$, and *Boolean* if $\mathsf{SVars}(P) = \emptyset$. For all $w \in \Sigma^*$, we say $P(w) = \mathtt{True}$ and $P(w) = \mathtt{False}$ instead of $P(w) = \{()\}$ and $P(w) = \emptyset$, respectively. Let P be a spanner and $w \in \Sigma^*$. A w-tuple $\mu \in P(w)$ is *hierarchical* if for all $x, y \in \mathsf{SVars}(P)$ at least one of the following holds:

1. The span $\mu(x)$ contains $\mu(y)$,
2. the span $\mu(y)$ contains $\mu(x)$, or
3. the spans $\mu(x)$ and $\mu(y)$ are disjoint.

A spanner P is *hierarchical* if, for every $w \in \Sigma^*$, every $\mu \in P(w)$ is hierarchical.

A spanner P is *total on* w if $P(w)$ contains all w-tuples over $\mathsf{SVars}(P)$. Let $Y \subset \mathsf{SVars}$ be a finite set of variables. The *universal spanner over* Y is denoted by Υ_Y. It is the unique spanner P' such that $\mathsf{SVars}(P') = Y$ and P' is total on every $w \in \Sigma^*$. Furthermore, a spanner P is *hierarchical total on* w if $P(w)$ is exactly the set of all hierarchical w-tuples over $\mathsf{SVars}(P)$; and the *universal hierarchical spanner* over a set Y is the unique spanner $\Upsilon_Y^{\mathbf{H}}$ that is hierarchical total on every $w \in \Sigma^*$.

For two spanners P_1 and P_2, we write $P_1 \subseteq P_2$ if $P_1(w) \subseteq P_2(w)$ for every $w \in \Sigma^*$, and $P_1 = P_2$ if $P_1(w) = P_2(w)$ for every $w \in \Sigma^*$.

Hence, a spanner can be understood as a function that maps a word w to a set of functions, each of which assigns spans of w to the variables of the spanner. As Boolean spanners are functions that map words to truth values, they can be interpreted as characteristic functions of languages. For every Boolean spanner P, we define the *language recognized by* P as $\mathcal{L}(P) := \{w \in \Sigma^* \mid P(w) = \texttt{True}\}$. We extend this to arbitrary spanners P by $\mathcal{L}(P) := \{w \in \Sigma^* \mid P(w) \neq \emptyset\}$.

▶ **Definition 8.** A *regex formula* is a regex α over Σ and $X := \mathsf{SVars}$ such that α does not contain any variable references, and for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = \gamma^*$, no subexpression of γ may be a variable binding.

In other words, a regex formula is a proper regular expression that is extended with variable binding operators, but these operators may not occur inside a Kleene star. We define $\mathsf{SVars}(\gamma) := \mathsf{Vars}(\gamma)$ for all regex formulas γ.

To define the semantics of regex formulas, we use the definition of parse trees for regexes, see Definition 2. Intuitively, the goal of this definition is that, each occurrence of a variable x in a γ-parse tree is matched to the corresponding span. Here, two problems can arise. Firstly, a variable might not occur in the parse tree; for example, when matching the regex formula $(x\{\mathtt{a}\} \vee \mathtt{bb})$ to the word $\mathtt{bb}$. Secondly, a variable might be defined too often, as e. g. in the regex formula $x\{\Sigma^+\} \cdot x\{\Sigma^+\}$. In order to avoid such problems, we introduce the notion of a functional regex formula.

▶ **Definition 9.** Let γ be a regex formula. We call γ *functional* if for every $w \in \Sigma^*$ and every γ-parse tree T_γ for w, each variable in $\mathsf{SVars}(\gamma)$ occurs in exactly one node label of T_γ. The class of all functional regex formulas is denoted by RGX.

As shown in Proposition 3.5 in Fagin et al. [7], functionality has a straightforward syntactic characterization: Basically, variables may not be redeclared, variables may not be used inside of Kleene stars, and if variables are used in a disjunction, each side of a disjunction has to contain exactly the same variables. Consider the following example:

▶ **Example 10.** The regex formula $\gamma_1 := (x\{\mathtt{a}\} \vee x\{\mathtt{b}\})$ is functional even though it contains two occurrences of variable definitions for x. There are just two γ_1-parse trees, both of which only contain one node labeled $(c, x\{c\})$, where $c \in \{\mathtt{a}, \mathtt{b}\}$. As a trivial case, even $\gamma_2 := x\{\emptyset\}$ is functional (as no γ_2-parse tree exists). Furthermore, the regex formulas $\gamma_3 := x\{(\mathtt{a} \vee \mathtt{b})^*\} \cdot x\{\mathtt{b}^+\}$ and $\gamma_4 := \mathtt{a}^* \vee x\{\mathtt{b}\}$ are not functional. Finally, $\gamma_5 := x\{\mathtt{a}\}^*$ is not a regex formula at all.

For functional regex formulas, we use parse trees to define the semantics:

▶ **Definition 11.** Let γ be a functional regex formula and let T be a γ-parse tree for a word $w \in \Sigma^*$. For every node v of T, the subtree that is rooted at v naturally maps to a span $p(v)$

of w. As γ is functional, for every $x \in \mathsf{SVars}(\gamma)$, exactly one node v_x of T has a label that contains x. We define $\mu^T \colon \mathsf{SVars}(\gamma) \to \mathsf{Spans}(w)$ by $\mu^T(x) := p(v_x)$. Each $\gamma \in \mathsf{RGX}$ defines a spanner $[\![\gamma]\!]$ by $[\![\gamma]\!](w) := \{\mu^T \mid T \text{ is a } \gamma\text{-parse tree for } w\}$ for each $w \in \Sigma^*$.

▶ **Example 12.** Assume that $\mathtt{a}, \mathtt{b} \in \Sigma$. We define the regex formula

$$\alpha := \Sigma^* \cdot x\big\{\mathtt{a} \cdot y\{\Sigma^*\} \cdot (z\{\mathtt{a}\} \vee z\{\mathtt{b}\})\big\} \cdot \Sigma^*.$$

Let $w := \mathtt{baaba}$. Then $[\![\alpha]\!](w)$ consists of the tuples $([2,4\rangle, [3,3\rangle, [3,4\rangle)$, $([2,5\rangle, [3,4\rangle, [4,5\rangle)$, $([2,6\rangle, [3,5\rangle, [5,6\rangle)$, $([3,5\rangle, [4,4\rangle, [4,5\rangle)$, $([3,6\rangle, [4,5\rangle, [5,6\rangle)$.

For every $w \in \Sigma^*$, a spanner P defines a (V, w)-relation $P(w)$. In order to construct more sophisticated spanners, we introduce spanner operators.

▶ **Definition 13.** Let P, P_1, P_2 be spanners and let $w \in \Sigma^*$. The algebraic operators *union*, *projection*, *natural join* and *selection* are defined as follows.

Union Two spanners P_1 and P_2 are *union compatible* if $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$, and their *union* $(P_1 \cup P_2)$ is defined by $\mathsf{SVars}(P_1 \cup P_2) := \mathsf{SVars}(P_1) \cup \mathsf{SVars}(P_2)$ and $(P_1 \cup P_2)(w) := P_1(w) \cup P_2(w)$ for every $w \in \Sigma^*$.

Projection Let $Y \subseteq \mathsf{SVars}(P)$. The *projection* $\pi_Y P$ is defined by $\mathsf{SVars}(\pi_Y P) := Y$ and $\pi_Y P(w) := P|_Y(w)$ for all $w \in \Sigma^*$, where $P|_Y(w)$ is the restriction of all w-tuples in $P(w)$ to Y.

Natural join Let $V_i := \mathsf{SVars}(P_i)$ for $i \in \{1, 2\}$. The *(natural) join* $(P_1 \bowtie P_2)$ of P_1 and P_2 is defined by $\mathsf{SVars}(P_1 \bowtie P_2) := \mathsf{SVars}(P_1) \cup \mathsf{SVars}(P_2)$ and, for all $w \in \Sigma^*$, $(P_1 \bowtie P_2)(w)$ is the set of all $(V_1 \cup V_2, w)$-tuples μ for which there exist (V_i, w)-tuples μ_i $(i \in \{1, 2\})$ with $\mu|_{V_1}(w) = \mu_1(w)$ and $\mu|_{V_2}(w) = \mu_2(w)$.

Selection Let $R \in (\Sigma^*)^k$ be a k-ary relation over Σ^*. The *selection operator* ζ^R is parameterized by k variables $x_1, \dots, x_k \in \mathsf{SVars}(P)$, written as $\zeta^R_{x_1,\dots,x_k}$. The *selection* $\zeta^R_{x_1,\dots,x_k} P$ is defined by $\mathsf{SVars}\big(\zeta^R_{x_1,\dots,x_k} P\big) := \mathsf{SVars}(P)$ and, for all $w \in \Sigma^*$, $\zeta^R_{x_1,\dots,x_k} P(w)$ is the set of all $\mu \in P(w)$ for which $\big(w_{\mu(x_1)}, \dots, w_{\mu(x_k)}\big) \in R$.

Like [7], we mostly consider the string equality selection operator $\zeta^=$. Hence, unless otherwise noted, the term "selection" refers to selection by the n-ary string equality relation. Note that unlike selection (which compares strings), join requires that the spans are identical.

Regarding the join of two spanners P_1 and P_2, $P_1 \bowtie P_2$ is equivalent to the intersection $P_1 \cap P_2$ if $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$, and to the Cartesian Product $P_1 \times P_2$ if $\mathsf{SVars}(P_1)$ and $\mathsf{SVars}(P_2)$ are disjoint. Hence, if applicable, we write $\cap$ and $\times$ instead of $\bowtie$.

For convenience, we may add and omit parentheses. We assume there is an order of precedence with projection and selection ranking over join ranking over union, e.g. we may write $\pi_Y \zeta^=_{x,y} P_1 \cup P_2 \bowtie P_3$ instead of $(\pi_Y \zeta^=_{x,y} P_1 \cup (P_2 \bowtie P_3))$, where projection and selection are applied to P_1, and the result is united with the join of P_2 and P_3.

▶ **Example 14.** Let $P_1 := \zeta^=_{x,y}[\![x\{\Sigma^*\} \cdot y\{\Sigma^*\}]\!]$ and $P_2 := \zeta^=_{x,y,z}[\![x\{\Sigma^*\} \cdot y\{\Sigma^*\} \cdot z\{\Sigma^*\}]\!]$. Then $\mathcal{L}(P_1) = \{ww \mid w \in \Sigma^*\}$, and the variables x and y always refer to the span of the first and second occurrence of w, respectively. Analogously, $\mathcal{L}(P_2) = \{www \mid w \in \Sigma^*\}$ (and z refers to the third occurrence of w). Assume that we want to construct a spanner for the language $\{w^n \mid w \in \Sigma^*, n \in \{2, 3\}\}$. As P_1 and P_2 are not union compatible, we cannot simply define $P_1 \cup P_2$. Union compatibility can be achieved by projecting P_2 to the set of common variables; i. e., $\pi_{\{x,y\}} P_2$.

▶ **Definition 15.** A *spanner algebra* is a finite set of spanner operators. If O is a spanner algebra, then $\mathsf{RGX}^{\mathsf{O}}$ denotes the set of all *spanner representations* that can be constructed

by (repeated) combination of the symbols for the operators from O with regex formulas from RGX. For each spanner representation of the form $o\rho$ (or $\rho_1\, o\, \rho_2$), where $o \in \mathsf{O}$, we define $[\![o\rho]\!] = o[\![\rho]\!]$ (and $[\![\rho_1\, o\, \rho_2]\!] = [\![\rho_1]\!]\, o\, [\![\rho_2]\!]$). Furthermore, $[\![\mathsf{RGX}^{\mathsf{O}}]\!]$ is the closure of $[\![\mathsf{RGX}]\!]$ under the spanner operators in O.

We define $\mathcal{L}(\rho) := \mathcal{L}([\![\rho]\!])$ for every spanner representation ρ. Fagin et al. [7] refer to $[\![\mathsf{RGX}]\!]$ as the class of *hierarchical regular spanners* and to $[\![\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}]\!]$ as the class of *regular spanners*. In addition to (hierarchical) regular spanners, Fagin et al. also introduced the so-called *core spanners*, which are obtained by combining regex formulas with the four algebraic operators projection, selection, union, and join – in other words, the class of core spanners is the class $[\![\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}]\!]$. Analogously, $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ is the class of *core spanner representations*.

3 Expressibility Results

3.1 Pattern Languages

We begin our examination of the expressive power of core spanners by comparing them to one of the simplest mechanisms with repetition operators:

▶ **Definition 16.** Let X be an infinite variable alphabet that is disjoint from Σ. A *pattern* is a word $\alpha \in (\Sigma \cup X)^+$ that generates the language $\mathcal{L}(\alpha) := \{\sigma(\alpha) \mid \sigma \text{ is a pattern substitution}\}$, where a *pattern substitution* is a homomorphism $\sigma\colon (\Sigma \cup X)^* \to \Sigma^*$ with $\sigma(a) = a$ for all $a \in \Sigma$. We denote the set of all variables in α by $\mathsf{Vars}(\alpha)$.

Intuitively, a pattern α generates exactly those words that can be obtained by replacing the variables in α with terminal words homomorphically (i. e., multiple occurrences of the same variable have to be replaced in the same way). This type of pattern languages is also called *erasing pattern language* (cf. Jiang et al. [19]).

▶ **Example 17.** Let $x, y \in X$ and $\mathtt{a}, \mathtt{b} \in \Sigma$. The patterns $\alpha := xx$ and $\beta := x\mathtt{a}y\mathtt{b}x$ generate the languages $\mathcal{L}(\alpha) = \{ww \mid w \in \Sigma^*\}$ and $\mathcal{L}(\beta) = \{v\mathtt{a}w\mathtt{b}v \mid v, w \in \Sigma^*\}$.

From every pattern α, we can straightforwardly construct a regex for $\mathcal{L}(\alpha)$. A similar observation holds for core spanners:

▶ **Theorem 18.** *Given a pattern α, we can compute in polynomial time a $\rho_\alpha \in \mathsf{RGX}^{\{\zeta^=\}}$ such that $\mathcal{L}(\rho_\alpha) = \mathcal{L}(\alpha)$.*

▶ **Example 19.** Let $x, y, z \in X$, $\mathtt{a}, \mathtt{b} \in \Sigma$, and define the pattern $\alpha := x\mathtt{a}yy\mathtt{b}xzx$. The construction in the proof of Theorem 18 leads to the spanner representation $\zeta^=_{x_1,x_2,x_3}\zeta^=_{y_1,y_2}\gamma$, where $\gamma = x_1\{\Sigma^*\} \cdot \mathtt{a} \cdot y_1\{\Sigma^*\} \cdot y_2\{\Sigma^*\} \cdot \mathtt{b} \cdot x_2\{\Sigma^*\} \cdot z_1\{\Sigma^*\} \cdot x_3\{\Sigma^*\}$.

While the construction in the proof of Theorem 18 is so easy that it might not seem noteworthy, it will prove quite useful: In contrast to their simple definition, many canonical decision problems for them are surprisingly hard. Via Theorem 18, the corresponding lower bounds also apply to spanners, as we discuss in Sections 4.1 and 4.2.

3.2 Word Equations and Existential Concatenation Formulas

In this section, we introduce word equations, which are equations of patterns (cf. Definition 16) and can be used to define languages and relations, cf. Karhumäki et al. [20]:

▶ **Definition 20.** A *word equation* is a pair $\eta = (\eta_L, \eta_R)$ of patterns η_L and η_R. A pattern substitution σ is a *solution* of η if $\sigma(\eta_L) = \sigma(\eta_R)$. We define $\mathsf{Vars}(\eta) := \mathsf{Vars}(\eta_L) \cup \mathsf{Vars}(\eta_R)$. For $k \geq 1$, a relation $R \subseteq (\Sigma^*)^k$ is defined by a word equation $\eta = (\eta_L, \eta_R)$ if there exist variables $x_1, \ldots, x_k \in \mathsf{Vars}(\eta)$ such that $R = \{(\sigma(x_1), \ldots, \sigma(x_k)) \mid \sigma \text{ is a solution of } \eta\}$.

We also write (η_L, η_R) as $\eta_L = \eta_R$. The following relations are well known examples of relations that are definable by word equations:

▶ **Definition 21.** Over Σ^*, we define relations $R_{\text{com}} := \{(x, y) \mid x, y \in \{u\}^* \text{ for some } u \in \Sigma^*\}$ and $R_{\text{cyc}} := \{(x, y) \mid x \text{ is a cyclic permutation of } y\}$.

As shown in Lothaire [22], the relation R_{com} is defined by the equation $xy = yx$, and R_{cyc} is defined by the equation $xz = zy$.

Let R be a k-ary string relation, and let C be a class of spanners. We say that R is *selectable* by C, if for every spanner $P \in C$ and every sequence of variables $\vec{x} = (x_1, \ldots, x_k)$ with $x_1, \ldots, x_k \in \mathsf{SVars}(P)$, the spanner $\zeta^R_{\vec{x}} P$ is also in C.

▶ **Proposition 22.** *The relations R_{com} and R_{cyc} are selectable by core spanners.*

In particular, this means that we can add $\zeta^{R_{\text{com}}}$ and $\zeta^{R_{\text{cyc}}}$ to core spanner representations, without leaving the class $[\![\mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}]\!]$.

▶ **Example 23.** Define $L_{\text{imp}} := \{w^n \mid w \in \Sigma^+, n \geq 2\}$ and $\rho := \zeta^{R_{\text{com}}}_{x,y}(x\{\Sigma^+\}y\{\Sigma^+\})$. Then $\mathcal{L}(\rho) = L_{\text{imp}}$.

This does not imply that R_{com} can be used to select relations like $R_{\text{pow}} := \{(x, x^n) \mid n \geq 0\}$. For example, if $x := \mathtt{abab}$, $(x, y) \in R_{\text{com}}$ holds for all $y \in \{\mathtt{ab}\}^*$. The authors conjecture that R_{pow} is not selectable by core spanners.

Furthermore, the spanner that is constructed for R_{com} in the proof of Proposition 22 is more complicated than the corresponding word equation $xy = yx$. In fact, we constructed both spanners not from the equations, but from a characterization of the solutions. This appears to be necessary, due the fact that spanners need to relate their variables to an input w, while word equations use their variables without such constrictions. We shall see in Theorem 28 further down that, if this restriction is kept in mind, core spanners can be used to simulate word equations.

Before we consider this topic further, we examine how word equations can simulate spanners, as this shall provide useful insights on some question of static analysis in Section 4.2. One drawback of word equations is that they are unable to express many comparatively simple regular languages; like A^* for any non-empty $A \subset \Sigma^*$ (cf. Karhumäki et al. [20]). In order to overcome this problem, we consider the following extension:

▶ **Definition 24.** Let $\eta = (\eta_L, \eta_R)$ be a word equation. A *regular constraints function*[1] is a function Cstr that maps each $x \in \mathsf{Vars}(\eta)$ to a regular language $\mathsf{Cstr}(x)$, where each of these languages is defined by a nondeterministic finite automaton. A solution σ of η is a *solution of η under constraints* Cstr if $\sigma(x) \in \mathsf{Cstr}(x)$ holds for every $x \in \mathsf{Vars}(\eta)$.

Hence, regular constraints restrict the possible substitutions of a variable x to a regular language $\mathsf{Cstr}(x)$.

A syntactic extension of word equations are *existential concatenation formulas*, which are obtained by extending word equations with $\vee$, $\wedge$, and existential quantification over

[1] While most existing literature uses the term *rational constraints*, we follow the terminology of [2].

variables. For example, R_{cyc} is expressed by the formula $\varphi_{\text{cyc}}(x, y) := \exists z\colon (xz = zy)$. Using appropriate coding techniques, one can transform every existential concatenation formula into an equivalent word equation (see Diekert [5]). In particular, this transformation is possible in polynomial time.

Like word equations, these formulas can be further extended by adding regular constraints. For each variable x and each nondeterministic finite automaton (NFA) A, the *(regular) constraint* $L_A(x)$ is satisfied for a solution σ if $\sigma(x) \in \mathcal{L}(A)$. We call the resulting formulas *existential concatenation formulas with regular constraints*, or $\mathsf{EC}^{\mathsf{reg}}$*-formulas*.

▶ **Example 25.** Let A be an NFA with $\mathcal{L}(A) = \{\mathtt{ab}^i\mathtt{a} \mid i \geq 1\}$, and define the $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi(x, y) := \exists z\colon (L_A(z) \wedge (\exists z_1, z_2\colon x = z_1 z z_2) \wedge (\exists z_1, z_2\colon y = z_1 z z_2))$.

Then φ expresses the relation of all (x, y) that have a common subword z from $\mathcal{L}(A)$.

Using the same techniques as for formulas without constraints, $\mathsf{EC}^{\mathsf{reg}}$-formulas can be transformed into equivalent word equations with regular constraints, and this construction is possible in polynomial time as well (cf. Diekert [5]). In order to express core spanners with $\mathsf{EC}^{\mathsf{reg}}$-formulas, we introduce the following definition:

▶ **Definition 26.** Let P be a core spanner with $\mathsf{SVars}(P) = \{x_1, \dots, x_n\}$, $n \geq 0$, and let $\varphi(x_w, x_1^P, x_1^C, \dots x_n^P, x_n^C)$ be an $\mathsf{EC}^{\mathsf{reg}}$-formula. We say that φ *realizes* P if, for all $w, w_1^P, w_1^C, \dots, w_n^P, w_n^C \in \Sigma^*$, $\varphi(w, w_1^P, w_1^C, \dots, w_n^P, w_n^C) = \mathtt{True}$ holds if and only if there is a $\mu \in P(w)$ with $w_k^P = w_{[1,i_k\rangle}$ and $w_k^C = w_{[i_k,j_k\rangle}$ for each $1 \leq k \leq n$, where $[i_k, j_k\rangle = \mu(x_k)$.

This definition uses the fact that spans are always defined in relation to a word w. Note that every span $[i, j\rangle \in \mathsf{Spans}(w)$ is characterized by the words $w_{[1,i\rangle}$ and $w_{[i,j\rangle}$. Hence, if $\mu \in [\![\rho]\!](w)$, the $\mathsf{EC}^{\mathsf{reg}}$-formula models $\mu(x_k) = [i_k, j_k\rangle$ by mapping x_w to w, x_k^P to $w_{[1,i_k\rangle}$, and x_k^C to $w_{[i_k,j_k\rangle}$. In the naming of the variables, C stands for *content*, and P for *prefix*. This allows us to model spanners in $\mathsf{EC}^{\mathsf{reg}}$-formulas:

▶ **Theorem 27.** *Given $\rho \in \mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}$ with $\mathsf{SVars}(\rho) = \{x_1, \dots, x_n\}$, $n \geq 0$, we can compute in polynomial time an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_\rho(x_w, x_1^P, x_1^C, \dots x_n^P, x_n^C)$ that realizes $[\![\rho]\!]$.*

As we shall see in Section 4.2, this result allows us to find upper bounds on two problems from the static analysis of spanners. We now examine how spanners can simulate word equations (and, thereby, also $\mathsf{EC}^{\mathsf{reg}}$-formulas). As discussed above, spanners need to relate their variables to an input word. Hence, we only state the following result, which is a weaker form of simulation than for the other direction:

▶ **Theorem 28.** *Every word equation $\eta = (\eta_L, \eta_R)$ with regular constraints Cstr can be converted computably into a $\rho \in \mathsf{RGX}^{\{\zeta^=, \bowtie\}}$ with $\mathsf{SVars}(\rho) \subseteq \mathsf{Vars}(\eta)$ such that for all $w \in \Sigma^*$, there is a solution σ of η under constraints Cstr with $w = \sigma(\eta_L) = \sigma(\eta_R)$ if and only if there is a $\mu \in [\![\rho]\!](w)$ with $\sigma(x) = w_{\mu(x)}$ for all $x \in \mathsf{Vars}(\eta)$.*

While this form of simulation is weaker (as w has to be present), it still shows that the constructed spanner is satisfiable if and only if the word equation (with constraints) is satisfiable; and computed (V, w)-relation encodes solutions of the equation.

▶ **Example 29.** Let $\mathtt{a}, \mathtt{b} \in \Sigma$ and define $\eta := (xy, yx)$ with $\mathsf{Cstr}(x) = \mathcal{L}(\mathtt{aab}^+)$, $\mathsf{Cstr}(y) = \Sigma^+$. The construction from the proof of Theorem 28 results in $\rho := \zeta^=_{x,x_2}\zeta^=_{y,y_2}(\hat{\eta}_L \times \hat{\eta}_R)$, where $\hat{\eta}_L := x\{\mathtt{aab}^+\} \cdot y\{\Sigma^+\}$ and $\hat{\eta}_R := y_2\{\Sigma^+\} \cdot x_2\{\mathtt{aab}^+\}$.

The only reason that this construction is not necessarily possible in polynomial time is that regular constraints are specified with NFAs, while core spanners use regular expressions, which can lead to an exponential increase in the size.

There is a similar construction that does not use the join operator: By adding new variables z_1, z_2, we can construct $\hat{\rho} := \zeta^{=}_{x,x_2}\zeta^{=}_{y,y_2}\zeta^{=}_{z_1,z_2}(z_1\{\hat{\eta}_L\}z_2\{\hat{\eta}_R\})$, which behaves almost like ρ; the only difference that the solution is encoded in $w = \sigma(\eta_L \cdot \eta_R)$, instead of $\sigma(\eta_L)$.

3.3 Regexes

As shown by Fagin et al. [7], there are languages that are recognized by regexes, but not by core spanners. In order to prove this, [7] introduced the so-called "uniform-0-chunk"-language L_{uzc}: Assuming $0, 1 \in \Sigma$, L_{uzc} is defined as the language of all $w = s_1 \cdot t \cdot s_2 \cdot t \cdots s_{n-1} \cdot t \cdot s_n$, where $n > 0$, $s_1, \ldots, s_n \in \{1\}^+$, and $t \in \{0\}^+$. Then $\mathcal{L}(\alpha_{\text{uzc}}) = L_{\text{uzc}}$ holds for the regex $\alpha_{\text{uzc}} := 1^+ \cdot x\{0^*\} \cdot (1^+ \cdot \&x)^* \cdot 1^+$, but no core spanner recognizes L_{uzc}.

Considering that the syntax of regex formulas does not allow the use of variables inside a Kleene star (or plus), this inexpressibility result might be considered expected, as α_{uzc} uses variable references inside of a Kleene plus. This raises the question whether regexes that restrict variables in a similar manner can still recognize languages that core spanners cannot. In order to examine this question, we define the following subclass of regexes:

▶ **Definition 30.** A regex α is *variable star-free* (short: *vstar-free*) if, for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = \gamma^*$, no subexpression of γ is a variable binding or a variable reference. We denote the class of all vstar-free regexes by vsfRX.

As we shall see in Theorem 36 below, every language that is recognized by a vstar-free regex is also recognized by a core spanner. While this observation might be considered not very surprising, its proof needs to deal with some technicalities. In particular, one needs to deal with expressions like $\alpha := x\{\Sigma^*\} \cdot (\&x \vee \&x\&x)$. A conversion in the spirit of Theorem 18 would need to replace the $\&x$ with distinct variables and ensure equality with selections; but as the disjunction contains subexpressions with distinct numbers of occurrences of $\&x$, we would not be able to ensure functionality of the resulting regex formula. We avoid these problems by working with the following syntactically restricted class of vstar-free regexes:

▶ **Definition 31.** An $\alpha \in \mathsf{vsfRX}$ is a *regex path* if, for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = (\gamma_1 \vee \gamma_2)$, no subexpression of γ_1 or γ_2 is a variable binding or a variable reference. We denote the class of all regex paths by RXP.

Intuitively, a regex path $\alpha \in \mathsf{RXP}$ can be understood as a concatenation $\alpha = \alpha_1 \cdots \alpha_n$, where each α_i is either a proper regular expression, a variable reference, or a variable binding of the form $\alpha_i = x\{\hat{\alpha}\}$, where $\hat{\alpha}$ is also a regex path. By "multiplying out" disjunctions that contain variables, we can convert every vstar-free regex into a disjunction of regex paths.

▶ **Lemma 32.** *Given $\alpha \in \mathsf{vsfRX}$, we can compute $\alpha_1, \ldots, \alpha_n \in \mathsf{RXP}$ with $\mathcal{L}(\alpha) = \bigcup_{i=1}^{n} \mathcal{L}(\alpha_i)$.*

▶ **Example 33.** Let $\alpha := x\{\Sigma^*\} \cdot (\&x \vee y\{\Sigma^*\}) \cdot (\&x \vee \&y)$. Multiplying out the disjunctions, we obtain regex paths $\alpha_1 = x\{\Sigma^*\} \cdot \&x \cdot \&x$, $\alpha_2 = x\{\Sigma^*\} \cdot y\{\Sigma^*\} \cdot \&x$, $\alpha_3 = x\{\Sigma^*\} \cdot \&x \cdot \&y$, and $\alpha_4 = x\{\Sigma^*\} \cdot y\{\Sigma^*\} \cdot \&y$. Then $\mathcal{L}(\alpha) = \bigcup_{i=1}^{4} \mathcal{L}(\alpha_i)$.

This transformation process might result in an exponential number of regex paths; but as efficiency is not of concern right now, this is not a problem. Each of these regex paths is then transformed into a functional regex formula:

▶ **Lemma 34.** *Given $\alpha \in \mathsf{RXP}$, we can be compute a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=}\}}$ with $\mathcal{L}(\rho) = \mathcal{L}(\alpha)$.*

▶ **Example 35.** Consider the regex path $\alpha := \&x \cdot x\{\Sigma^* \cdot y\{\Sigma^*\}\} \cdot \&x \cdot \&y \cdot y\{\Sigma^*\} \cdot \&x \cdot \&y$. The construction from the proof of Lemma 34 leads to the equivalent regex path $\gamma := \varepsilon \cdot x\{\Sigma^* \cdot y\{\Sigma^*\}\} \cdot \&x \cdot \&y \cdot \hat{y}\{\Sigma^*\} \cdot \&x \cdot \&\hat{y}$, from which we derive the functional regex formula

$$\delta := x\,\{\Sigma^* y\{\Sigma^*\}\}\, x_1\{\Sigma^*\} y_1\{\Sigma^*\} \hat{y}\{\Sigma^*\} x_2\{\Sigma^*\} \hat{y}_1\{\Sigma^*\},$$

which we use in the spanner representation $\rho := \pi_\emptyset \zeta^=_{x,x_1,x_2} \zeta^=_{y,y_1} \zeta^=_{\hat{y},\hat{y}_1} \delta$. Then $\mathcal{L}(\alpha) = \mathcal{L}(\rho)$.

As these spanner representations are Boolean, they are also union compatible. Hence, we can now combine Lemma 32 and Lemma 34 to observe the following.

▶ **Theorem 36.** *Given $\alpha \in$ vsfRX, we can compute a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup\}}$ with $\mathcal{L}(\rho) = \mathcal{L}(\alpha)$.*

In Section 4.2, we use Theorem 36 together with the undecidability results from [11] to obtain multiple lower bounds for static analysis problems. Theorem 36 also raises the question whether every language that is recognized by a core spanner is also recognized by a vstar-free regular expression. As we have already seen in Example 23, it is possible to express the language $L_{\text{imp}} := \{w^n \mid w \in \Sigma^+, n \geq 2\}$ with core spanners. Hence, under certain conditions, core spanners can simulate constructions like $(\&x)^*$.

While L_{imp} might seem to be an obvious witness that separates the classes of languages that are recognized by core spanners and by vstar-free regexes, proving this appears to be quite involved. Instead, we consider a related language, which allows us to use the following tool:

▶ **Definition 37.** Let $k \in \mathbb{N}_{>0}$. We call a set $A \subseteq \mathbb{N}^k$ *linear* if there exist an $r \geq 0$ and $m_0, \ldots, m_r \in \mathbb{N}^k$ with $A = \{m_0 + m_1 i_1 + m_2 i_2 + \cdots + m_r i_r \mid i_1, i_2, \ldots, i_r \in \mathbb{N}\}$. A set $A \subseteq \mathbb{N}^k$ is *semi-linear* if it is a finite union of linear sets. Assume Σ is ordered; i. e., $\Sigma = \{a_1, a_2 \ldots, a_k\}$. The *Parikh map* $\Psi\colon \Sigma^* \to \mathbb{N}^k$ is defined by $\Psi(w) := (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_k})$, and is extended to languages by $\Psi(L) := \{\Psi(w) \mid w \in L\}$. We call L *semi-linear* if $\Psi(L)$ is semi-linear.

According to Parikh's Theorem [24], every context-free language is semi-linear. Moreover, as shown by Ginsburg and Spanier [15], a set is semi-linear if and only if it is definable in Presburger arithmetic. Building on this, we state the following:

▶ **Theorem 38.** *For every $\alpha \in$ vsfRX, the language $\mathcal{L}(\alpha)$ is semi-linear.*

We use Theorem 38 to separate the classes of languages that are recognized by core spanners and by vstar-free regexes:

▶ **Lemma 39.** *Let $L_{\text{nsl}} := \{(\mathtt{ab}^m)^n \mid m, n \geq 2\}$ and $\rho := \zeta^{R_{\text{com}}}_{x,y}(x\{\mathtt{abb}^+\}y\{\Sigma^+\})$ for $\Sigma := \{\mathtt{a}, \mathtt{b}\}$. Then $L_{\text{nsl}} = \mathcal{L}(\rho)$, but there is no $\alpha \in$ vsfRX with $\mathcal{L}(\alpha) = L_{\text{nsl}}$.*

We do not need the join operator to define non-semi-linear languages: Consider the core spanner representation ρ from Example 29 with $\mathcal{L}(\rho) = L_{\text{nsl}}$. If we construct $\hat{\rho}$ as explained below that example, we obtain $\mathcal{L}(\hat{\rho}) = \{ww \mid w \in L_{\text{nsl}}\}$, which is also not semi-linear.

It is worth pointing out Lemma 39 does not resolve the open question from [7] whether there is a language that is recognized by a core spanner, but not by a regex, as Theorem 38 only applies to vstar-free regexes. We have already seen languages that are not semi-linear, but are recognized by regexes: The language L_{nsl} is recognized by $\alpha_{\text{nsl}} := x\{\mathtt{abb}^+\}\&x^+$; and a similar approach is used for the following language (which we already met in Example 4):

▶ **Example 40.** Let $\Sigma := \{\mathtt{a}\}$, and define the language $L_{\text{npr}} := \{\mathtt{a}^{mn} \mid m, n \geq 2\}$. In other words, L_{npr} is the language of all words $\mathtt{a}^i$ with $i \geq 4$ such that i is not a prime number. Let $\alpha_{\text{npr}} := x\{\mathtt{aa}^+\} \cdot (\&x)^+$. Then $\mathcal{L}(\alpha_{\text{npr}}) = L_{\text{npr}}$.

While L_{nsl} and L_{npr} are defined by very similar regexes, the latter cannot be recognized by core spanners. In order to show this with a semi-linearity argument, we observe:

▶ **Theorem 41.** *Let $|\Sigma| = 1$ and let P be a core spanner over Σ. Then $\mathcal{L}(P)$ is semi-linear.*

Apart from the observation that L_{npr} from Example 40 is not recognized by core spanners, Theorem 41 also allows us to conclude that on unary alphabets, core spanners recognize exactly the class of regular languages (which, on unary alphabets, is identical to the class of context-free languages).

4 Decision Problems

4.1 Spanner Evaluation

We first examine the *combined complexity* of the evaluation problem for core spanners, the problem CSp-Eval: Given a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$, a $w \in \Sigma^*$, and a $(\mathsf{SVars}(\rho), w)$-tuple μ, is $\mu \in [\![\rho]\!](w)$? In order to prove lower bounds for this problem, we consider the membership problem for pattern languages: Given a pattern α and a word w, decide whether $w \in \mathcal{L}(\alpha)$. As shown by Jiang et al. [19], this problem is NP-complete. Due to Theorem 18, we observe the following (the proof of NP-membership is straightforward).

▶ **Theorem 42.** *CSp-Eval is NP-complete, even if restricted to $\mathsf{RGX}^{\{\pi,\zeta^{=}\}}$.*

The question arises whether there are natural restrictions to CSp-Eval that make this problem tractable. It appears that any subclass of the core spanners that extends regular spanners in a meaningful way while having a tractable evaluation problem can not be allowed to recognize the full class of pattern languages.

For pattern languages, it was shown by Ibarra et al. [18] that bounding the number of variables in the pattern leads to an algorithm for the membership problem with a running time that is polynomial, although in $\mathcal{O}(n^k)$ (where n is the length of the word w, and k the number of variables). From a parameterized complexity point of view, this is usually not considered satisfactory. In fact, it was first observed by Stephan et al. [26] that the membership problem for pattern languages is $W[1]$-complete if the number of variable occurrences (not of variables) is used as a parameter. As the number of variable occurrences in a pattern corresponds to the number of variables in an equivalent spanner, this implies that using the number of variables in a spanner as parameter leads to $W[1]$-hardness for this parameter of CSp-Eval.

Fernau and Schmid [9] and Fernau et al. [10] discuss various other potential restrictions to pattern languages that still do not lead to tractability (among these a bound on the length of the replacement of each variable, which corresponds to a bound on the length of spans). On the other hand, Reidenbach and Schmid [25] and Fernau et al. [8] examine parameters for patterns that make the membership problem tractable. While this does not directly translate to spanners, the authors consider these directions promising for further research.

We also consider the *data complexity* of the evaluation problem for core spanners. For every core spanner representation ρ over Σ, we define the decision problem CSp-Eval(ρ): Given a $w \in \Sigma^*$ and a w-tuple μ, is $\mu \in [\![\rho]\!](w)$? Using a slight variation of the proof of Theorem 42, we obtain the following.

▶ **Theorem 43.** *For every $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$, CSp-Eval($\rho$) is in NL.*

4.2 Static Analysis

We consider the following common decision problems for core spanner representations, where the input is $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$ or $\rho_1, \rho_2 \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$:

1. CSp-Sat: Is $[\![\rho]\!](w) \neq \emptyset$ for some $w \in \Sigma^*$?
2. CSp-Hierarchicality: Is $[\![\rho]\!]$ hierarchical?
3. CSp-Universality: Is $[\![\rho]\!] = \Upsilon_{\mathsf{SVars}(\rho)}$?
4. CSp-Equivalence: Is $[\![\rho_1]\!] = [\![\rho_2]\!]$?
5. CSp-Containment: Is $[\![\rho_1]\!] \subseteq [\![\rho_2]\!]$?
6. CSp-Regularity: Is $[\![\rho]\!] \in [\![\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}]\!]$?

We approach the first two of these problems by using Theorem 27 to convert core spanner representations to $\mathsf{EC}^{\mathsf{reg}}$-formulas, for which satisfiability is in PSPACE (cf. Diekert [5]). Hence, we observe:

▶ **Theorem 44.** *CSp-Sat is* PSPACE*-complete, even if restricted to* $\mathsf{RGX}^{\{\zeta^=\}}$.

For the lower bound, the proof of Theorem 44 uses the PSPACE-hardness of the intersection emptiness problem for regular expressions. But even if the variables in the regex formulas were only bound to Σ^*, it follows from Theorem 28 that this problem would still be at least as hard as the satisfiability problem for word equations without constraints (cf. Diekert [5]).

Furthermore, it is possible to use $\mathsf{EC}^{\mathsf{reg}}$-formulas to express a violation of the criteria for hierarchicality. This allows us to state the following result:

▶ **Theorem 45.** *CSp-Hierarchicality is* PSPACE*-complete, even if restricted to* $\mathsf{RGX}^{\{\zeta^=,\bowtie\}}$.

For the remaining problems, we use Theorem 36, and the fact that the undecidability results from Freydenberger [11] also hold for vstar-free regexes:

▶ **Theorem 46.** *CSp-Universality and CSp-Equivalence are not semi-decidable, and CSp-Regularity is neither semi-decidable, nor co-semi-decidable. This holds even if the input is restricted to* $\mathsf{RGX}^{\{\pi,\zeta^=,\cup\}}$.

As the proof of Theorem 46 relies only on Boolean spanners, the decidability status of CSp-Regularity does not change if the problem asks for hierarchical regularity (i. e., membership in $[\![\mathsf{RGX}]\!]$) instead of regularity, as the two classes coincide for Boolean spanners. Likewise, CSp-Universality remains not semi-decidable if one replaces $\Upsilon_{\mathsf{SVars}(\rho)}$ with $\Upsilon^{\mathbf{H}}_{\mathsf{SVars}(\rho)}$.

In the construction from this proof, variables are only bound to a language a^+. Hence, the same undecidability results hold for spanners that use selections by equal length relation, instead of the string equality relation. While the proof builds on regexes $\alpha_{\mathcal{X}}$ that use only a single variable x, the resulting core spanners use an unbounded amount variables, as every occurrence of a variable reference $\&x$ in a regex path is converted to a spanner variable x_i. But undecidability remains even if we bound the number of variables in the spanners, as the $\alpha_{\mathcal{X}}$ can be modified to use only a bounded number of variable references (see Section 4.1 in [11]). Theorem 46 also implies that CSp-Containment is not semi-decidable. This holds even for a more restricted class of spanners:

▶ **Theorem 47.** *CSp-Containment is not semi-decidable, even if restricted to* $\mathsf{RGX}^{\{\pi,\zeta^=\}}$.

As shown by Bremer and Freydenberger [3], the inclusion problem for pattern languages remains undecidable if the number of variables in the patterns is bounded. In fact, that proof constructs patterns where even the number of variable occurrences is bounded. Therefore, CSp-Containment is not semi-decidable even if restricted to representations from $\mathsf{RGX}^{\{\pi,\zeta^=\}}$ with a bounded number of variables. It is a hard open question whether the equivalence problem for pattern languages is decidable (cf. Ohlebusch and Ukkonen [23], Freydenberger and Reidenbach [12]). Undecidability of this problem would imply undecidability of CSp-Equivalence, even if restricted to representations from $\mathsf{RGX}^{\{\pi,\zeta^=\}}$.

4.2.1 Minimization and Relative Succinctness

In order to address the minimization of spanner representations, we first formalize the notion of the size or complexity of a spanner representation. Even for proper regular expressions, there are various different definitions of size, see e. g. Holzer and Kutrib [17], and there might be convincing reasons to add additional weight to the number of variables or other parameters. As we shall see, these distinctions do not affect the negative results that we prove further down. Hence, instead of defining a single fixed notion of size, we use the following general definition of complexity measures from Kutrib [21]:

▶ **Definition 48.** Let SR be a class of spanner representations. A *complexity measure* for SR is a recursive function $c : \mathsf{SR} \to \mathbb{N}$ such that for each Σ, the set of all $\rho \in \mathsf{SR}$ that represent spanners over Σ can be computably enumerated in order of increasing $c(\rho)$, and does not contain infinitely many $\rho \in \mathsf{SR}$ with the same value $c(\rho)$.

By *recursive*, we mean a function that is total and computable. Definition 48 is general enough to include all notions of complexity that take into account that descriptions are commonly encoded with a finite number of distinct symbols, and that it should be decidable if a word over these symbols is a valid encoding from SR. Regardless of the chosen complexity measure, computable minimization of core spanners is impossible:

▶ **Theorem 49.** *Let c be a complexity measure for $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$. There is no algorithm that, given a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, computes an equivalent $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ that is c-minimal.*

In addition to regex formulas, Fagin et al. [7] also define spanner representations that are based on so-called vset- and vstk-automata (denoted by $\mathsf{VA}_{\mathsf{set}}$ and $\mathsf{VA}_{\mathsf{stk}}$) and extended with the same spanner operators; and they compare the expressive power of these spanner representations to $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ and its subclasses. Without going into details, we note that their equivalence proofs use computable conversions between the models. Hence, Theorem 49 also applies to those spanner representations from [7] that can express core spanners, like ${\mathsf{VA}_{\mathsf{stk}}}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ and ${\mathsf{VA}_{\mathsf{set}}}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, and it implies that an algorithm that converts from one of these classes of representations to another cannot guarantee that its result is minimal.

Using a technique by Hartmanis [16], we can use the fact that CSp-Regularity is not co-semi-decidable to compare the relative succinctness of regular and core spanner representations:

▶ **Theorem 50.** *Let c_1, c_2 be complexity measures for $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ and $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, respectively. For every recursive function $f : \mathbb{N} \to \mathbb{N}$, there exists a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ such that $[\![\rho]\!] \in [\![\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}]\!]$, but $c_1(\hat{\rho}) > f(c_2(\rho))$ holds for every $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ with $[\![\hat{\rho}]\!] = [\![\rho]\!]$.*

Hence, the blowup from $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ to $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ is not bounded by a recursive function. As above, we can replace each of this classes with a class with the same expressive power; for example, we can replace $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ with ${\mathsf{VA}_{\mathsf{stk}}}^{\{\pi,\cup,\bowtie\}}$, $\mathsf{VA}_{\mathsf{set}}$, or ${\mathsf{VA}_{\mathsf{set}}}^{\{\pi,\cup,\bowtie\}}$ (or, as the proof uses Boolean spanners, RGX or $\mathsf{VA}_{\mathsf{stk}}$, or any class between those).

We also consider the relative succinctness of representations of core spanners and representations of their complements. For every spanner P, we define its *complement* $\mathsf{C}(P) := \Upsilon_{\mathsf{SVars}(P)} \setminus P$, and its *hierarchical complement* $\mathsf{C}^{\mathbf{H}}(P) := \Upsilon^{\mathbf{H}}_{\mathsf{SVars}(P)} \setminus P$.

▶ **Theorem 51.** *Let c be a complexity measure for $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$. For every recursive function $f : \mathbb{N} \to \mathbb{N}$, there exists a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ such that $\mathsf{C}([\![\rho]\!]) \in [\![\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}]\!]$, but $c(\rho) > f(c(\hat{\rho}))$ holds for every $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ with $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$. This also holds if we consider $\mathsf{C}^{\mathbf{H}}$ instead of C.*

In other words, there are core spanners where the (hierarchical) complement is also a core spanner, but the blowup between their representations is not bounded by any recursive function. Again, this holds for the other classes of representations as well.

This result has consequences to an open question of Fagin et al. One of the central tools in [7] is the core-simplification-lemma, which states that every core spanner is definable by an expression of the form $\pi_V SA$, where A is a vset-automaton, $V \subseteq \mathsf{SVars}(A)$, and S is a sequence of selections $\zeta^{=}_{x,y}$ for $x, y \in \mathsf{SVars}(A)$.

In addition to core spanners, Fagin et al. also discuss adding a set difference operator $\setminus$, and ask "whether we can find a simple form, in the spirit of the core-simplification lemma, when adding difference to the representation of core spanners". It is a direct consequence of Theorem 51 that such a simple representation, if it exists, cannot be obtained computably, as reducing the number of difference operators can lead to a non-recursive blowup. While this observation does not prove that such a simple form does not exist, it suggests that any proof of its existence should be expected to be non-constructive.

5 Conclusions and Further Work

In Section 3, we have seen that core spanners can express languages that are defined by patterns or by vstar-free regexes. We used this in Section 4 to derive various lower bounds on decision problems, even for subclasses of core spanner representations. Note that in most of the cases, these lower bounds do not require the join operator, and mostly rely on the string equality selection. This can be interpreted as a sign that string equality (or repetition) is an expensive operator, in particular as similar results have been observed for related models (e. g., [1, 11, 13]).

On a more positive note, there is reason to hope that these connections can be beneficial for spanners: There is recent work on restricted classes of pattern languages with an efficient membership problem (e. g., [9, 25]), which could lead to subclasses of spanners that can be evaluated more efficiently. Furthermore, as Theorems 27 and 28 show, core spanners and word equations with regular constraints are closely related. Recent work on word equations has also considered tasks like enumerating all solutions of an equation. The employed compression techniques (cf. [5]) might also be used to improve the evaluation of core spanners. In particular, the $\mathsf{EC}^{\mathsf{reg}}$-formulas that are constructed in the proof of Theorem 27 have the special property that there is a variable x_w (for w), and for every solution σ and every variable x, $\sigma(x)$ is a subword of $\sigma(x_w)$. It remains to be seen whether this restriction leads to favorable lower bounds.

Also note that conversion from vstar-free regular expressions to core spanner representations that is used for Theorem 36 can lead to an exponential increase in size. If this size blowup cannot be avoided, allowing vstar-free regexes as primitive spanner representations might be useful as syntactic sugar.

Finally, while we only mentioned this explicitly in Section 4.2.1, note that most of the other results in this paper can also be directly converted to the appropriate spanner representations that use vset- and vstk-automata from [7].

Acknowledgements. We thank Florin Manea for his suggestion to use word equations with regular constraints, and Thomas Zeume for reporting a list of typos. We also thank the anonymous reviewers for their feedback.

References

1 P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM T. Database Syst.*, 37(4):31, 2012.

2 P. Barceló and P. Muñoz. Graph logics with rational relations: the role of word combinatorics. In *Proc. CSL-LICS 2014*, 2014.

3 J. Bremer and D. D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. *Inform. Comput.*, 220–221:15–43, 2012.

4 C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *Int. J. Found. Comput. Sci.*, 14:1007–1018, 2003.

5 V. Diekert. Makanin's Algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12, pages 387–442. Cambridge University Press, 2002.

6 R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Cleaning inconsistencies in information extraction via prioritized repairs. In *Proc. PODS 2014*, 2014.

7 R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015.

8 H. Fernau, F. Manea, R. Mercas, and M. L. Schmid. Pattern matching with variables: Fast algorithms and new hardness results. In *Proc. STACS 2015*, 2015.

9 H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Inf. Comput.*, 242:287–305, 2015.

10 H. Fernau, M. L. Schmid, and Y. Villanger. On the parameterised complexity of string morphism problems. *Theory Comput. Sys.*, 2015.

11 D. D. Freydenberger. Extended regular expressions: Succinctness and decidability. *Theory Comput. Sys.*, 53(2):159–193, 2013.

12 D. D. Freydenberger and D. Reidenbach. Bad news on decision problems for patterns. *Inform. Comput.*, 208(1):83–96, 2010.

13 D. D. Freydenberger and N. Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. *J. Comput. Syst. Sci.*, 79(6):892–909, 2013.

14 J. E. F. Friedl. *Mastering Regular Expressions*. O'Reilly Media, 3rd edition, 2006.

15 S. Ginsburg and E. Spanier. Semigroups, presburger formulas, and languages. *Pac. J. Math.*, 16(2):285–296, 1966.

16 J. Hartmanis. On Gödel speed-up and succinctness of language representations. *Theor. Comput. Sci.*, 26(3):335–342, 1983.

17 M. Holzer and M. Kutrib. Descriptional complexity–an introductory survey. *Scientific Applications of Language Methods*, 2:1–58, 2010.

18 O. H. Ibarra, T.-C. Pong, and S. M. Sohn. A note on parsing pattern languages. *Pattern Recogn. Lett.*, 16(2):179–182, 1995.

19 T. Jiang, E. Kinber, A. Salomaa, K. Salomaa, and S. Yu. Pattern languages with and without erasing. *Int. J. Comput. Math.*, 50:147–163, 1994.

20 J. Karhumäki, F. Mignosi, and W. Plandowski. The expressibility of languages and relations by word equations. *J. ACM*, 47(3):483–505, 2000.

21 M. Kutrib. The phenomenon of non-recursive trade-offs. *Int. J. Found. Comput. Sci.*, 16(5):957–973, 2005.

22 M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.

23 E. Ohlebusch and E. Ukkonen. On the equivalence problem for E-pattern languages. *Theor. Comput. Sci.*, 186:231–248, 1997.

24 R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

25 D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. *Inform. Comput.*, 239:87–99, 2014.

26 F. Stephan, R. Yoshinaka, and T. Zeugmann. On the parameterised complexity of learning patterns. In *Proc. ISCIS 2011*, 2011.

Algorithms for Provisioning Queries and Analytics*

Sepehr Assadi[†1], Sanjeev Khanna[†2], Yang Li[†3], and Val Tannen[‡4]

1 **University of Pennsylvania, Philadelphia, PA, USA**
sassadi@cis.upenn.edu
2 **University of Pennsylvania, Philadelphia, PA, USA**
sanjeev@cis.upenn.edu
3 **University of Pennsylvania, Philadelphia, PA, USA**
yangli2@cis.upenn.edu
4 **University of Pennsylvania, Philadelphia, PA, USA**
val@cis.upenn.edu

Abstract

Provisioning is a technique for avoiding repeated expensive computations in *what-if analysis*. Given a query, an analyst formulates k *hypotheticals*, each retaining some of the tuples of a database instance, *possibly overlapping*, and she wishes to answer the query under *scenarios*, where a scenario is defined by a subset of the hypotheticals that are "turned on". We say that a query admits *compact* provisioning if given any database instance and any k hypotheticals, one can create a poly-size (in k) *sketch* that can then be used to answer the query under any of the 2^k possible scenarios without accessing the original instance.

In this paper, we focus on provisioning complex queries that combine relational algebra (the logical component), grouping, and statistics/analytics (the numerical component). We first show that queries that compute quantiles or linear regression (as well as simpler queries that compute count and sum/average of positive values) can be compactly provisioned to provide (multiplicative) *approximate* answers to an arbitrary precision. In contrast, *exact* provisioning for each of these statistics requires the sketch size to be exponential in k. We then establish that for any complex query whose logical component is a *positive* relational algebra query, as long as the numerical component can be compactly provisioned, the complex query itself can be compactly provisioned. On the other hand, introducing negation or recursion in the logical component again requires the sketch size to be exponential in k. While our positive results use algorithms that do not access the original instance after a scenario is known, we prove our lower bounds even for the case when, knowing the scenario, limited access to the instance is allowed.

1998 ACM Subject Classification H.2.8 Database Applications

Keywords and phrases What-if Analysis, Provisioning, Data Compression, Approximate Query Answering

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.18

1 Introduction

"What if analysis" is a common technique for investigating the impact of decisions on outcomes in science or business. It almost always involves a data analytics computation. Nowadays

* The full version of the paper can be found at [3].
† Supported in part by National Science Foundation grants CCF-1116961, CCF-1552909, and IIS-1447470 and an Adobe research award.
‡ Supported in part by National Science Foundation grants IIS-1217798 and IIS-1302212.

such a computation typically processes very large amounts of data and thus may be expensive to perform, especially repeatedly. An analyst is interested in exploring the computational impact of multiple *scenarios* that assume modifications of the input to the analysis problem. Our general aim is to avoid repeating expensive computations for each scenario. For a given problem, and starting from a given set of potential scenarios, we wish to perform just *one* possibly expensive computation producing a small *sketch* (i.e., a compressed representation of the input) such that the answer for any of the given scenarios can be derived rapidly from the sketch, without accessing the original (typically very large) input. We say that the sketch is "provisioned" to deal with the problem under any of the scenarios and following [13], we call the whole approach *provisioning*. Again, the goal of provisioning is to allow an analyst to efficiently explore a multitude of scenarios, using only the sketch and thus avoiding expensive recomputations for each scenario.

In this paper, we apply the provisioning approach to queries that perform *in-database analytics* [25][1]. These are queries that combine *logical* components (relational algebra and Datalog), grouping, and *numerical* components (e.g., aggregates, quantiles and linear regression). Other analytics are discussed under further work.

Abstracting away any data integration/federation, we will assume that the inputs are relational instances and that the scenarios are defined by a set of *hypotheticals*. We further assume that each hypothetical indicates the fact that certain tuples of an input instance are *retained* (other semantics for hypotheticals are discussed under further work).

A scenario consists of turning on/off each of the hypotheticals. Applying a scenario to an input instance therefore means keeping only the tuples retained by at least one of the hypotheticals that are turned on. Thus, a trivial sketch can be obtained by applying each scenario to the input, solving the problem for each such modified input and collecting the answers into the sketch. However, with k hypotheticals, there are *exponentially* (in k) many scenarios. Hence, even with a moderate number of hypotheticals, the size of the sketch could be enormous. Therefore, as part of the statement of our problem we will aim to provision a query by an algorithm that maps each (large) input instance to a *compact* (essentially size $\text{poly}(k)$) sketch.

▶ **Example 1.** Suppose a large retailer has many and diverse sales venues (e.g., its own stores, its own web site, through multiple other stores, and through multiple other web retailers). An analyst working for the retailer is interested in learning, for each product in, say, "Electronics", a regression model for the way in which the *revenue* from the product depends on both a sales venue's *reputation* (assume a numerical score) and a sales venue *commission* (in %; 0% if own store). Moreover, the analyst wants to ignore products with small sales volume unless they have a large MSRP (manufacturer's suggested retail price). Usually there is a large (possibly distributed/federated) database that captures enough information to allow the computation of such an analytic query. For simplicity we assume in this example that the revenue for each product ID and each sales venue is in one table and thus we have the following query with a self-explanatory schema:

```
SELECT x.ProdID, LIN_REG(x.Revenue, z.Reputation, z.Commission) AS (B,A1,A2)
FROM  RevenueByProductAndVenue x
INNER JOIN Products y ON x.ProdID=y.ProdID
```

[1] In practice, the MADlib project [29] has been one of the pioneers for in-database analytics, primarily in collaboration with Greenplum DB [21]. By now, major RDBMS products such as IBM DB2, MS SQL Server, and Oracle DB already offer the ability to combine extensive analytics with SQL queries.

```
INNER JOIN SalesVenues z ON x.VenueID=z.VenueID
WHERE y.ProdCategory="Electronics" AND (x.Volume>100 OR y.MSRP>1000)
GROUP BY x.ProdID
```

The syntax for treating linear regression as a multiple-column-aggregate is simplified for illustration purposes in this example. Here the values under the attributes `B,A1,A2` denote, for each `ProdID`, the coefficients of the linear regression model that is learned, i.e., `Revenue = B + A1*Reputation + A2*Commission`.

A desirable what-if analysis for this query may involve hypotheticals such as retaining certain venue types, retaining certain venues with specific sales tax properties, retaining certain product types (within the specified category, e.g., tablets), and many others. Each of these hypotheticals can in fact be implemented as selections on one or more of the tables in the query (assuming that the schema includes the appropriate information). However, combining hypotheticals into scenarios is problematic. The hypotheticals overlap and thus cannot be separated. With 10 (say) hypotheticals there will be $2^{10} = 1024$ (in practice at least hundreds) of regression models of interest for each product. Performing a lengthy computation for each one of these models is in total very onerous. Instead, we can *provision* the what-if analysis of this query as the query in this example falls within the class covered by our positive results.

Our results. Our goal is to characterize the feasibility of provisioning with sketches of *compact* size (see Section 2 for a formal definition) for a practical class of *complex queries* that consist of a *logical* component (relational algebra or Datalog), followed by a *grouping* component, and then by a *numerical* component (aggregate/analytic) that is applied to each group (a more detailed definition is given in Section 5).

The main challenge that we address, and the part where our main contribution lies, is the design of compact provisioning schemes for numerical queries, specifically linear (ℓ_2) regression and quantiles. Together with the usual count, sum and average, these are defined in Section 4 as queries that take a set of numbers or of tuples as input and return a number or a tuple of constant width as output. It turns out that if we expect exact answers, then none of these queries can be compactly provisioned. However, we show that compact provisioning schemes indeed exist for all of them if we relax the objective to computing near-exact answers (see Section 2 for a formal definition). The following theorem summarizes our results for numerical queries (see Section 4):

▶ **Theorem 2** (Informal). *The* quantiles, linear (ℓ_2) regression, count, and sum/average (of positive numbers) queries *can be compactly provisioned to provide (multiplicative) approximate answers to an arbitrary precision, while their exact provisioning requires the sketch size to be exponential in the number of hypotheticals.*

Our results on provisioning numerical queries can then be used for complex queries as the following theorem summarizes (see Section 5):

▶ **Theorem 3** (Informal). *Any complex query whose logical component is a* positive relational algebra query *can be compactly provisioned to provide an approximate answer to an arbitrary precision as long as its numerical component can be compactly provisioned for the same precision. On the other hand, introducing* negation *or* recursion *in the logical component, requires the sketch size to be exponential in the number of hypotheticals.*

Our techniques. At a high-level, our approach for compact provisioning can be described as follows. We start by building a sub-sketch for each hypothetical by focusing solely on

the retained tuples of each hypothetical individually. We then examine these sub-sketches against each other and collect additional information from the original input to summarize the effect of appearance of other hypotheticals to each already computed sub-sketch. The first step usually involves using well-known (and properly adjusted) sampling or sketching techniques, while the second step, which is where we concentrate the bulk of our efforts, is responsible for gathering the information required for combining the sketches and specifically dealing with overlapping hypotheticals. Given a scenario, we answer the query by fetching the corresponding sub-sketches and merging them together; the result is a new sketch that act as sketch for the input consist of the *union* of the hypotheticals.

We prove our lower bounds by first identifying a central problem, i.e., the Coverage problem (see Problem 8), with provably large space requirement for any provisioning scheme, and then use reductions from this problem to establish the lower bound for other queries of interest. The space requirement of the Coverage problem itself is proven using simple tools from information theory.

Comparison with existing work. Our techniques for compact provisioning share some similarities with those used in data streaming and in the distributed computation model of [12, 35], and in particular *linear sketching*, which corresponds to applying a linear transformation to the input data to obtain the sketch. However, due to overlap in the input, our sketches are required to to be composable with the *union* operation (instead of the *addition* operation obtained by linear sketches) and hence linear sketching techniques are not directly applicable.

Dealing with duplicates in the input (similar to the overlapping hypotheticals) has also been considered in the streaming and distributed computation models (see, e.g., [10, 7]), which consider sketches that are "duplicate-resilient". Indeed, for simple queries like count, a direct application of these sketches is sufficient for compact provisioning (see Section 4.1). We also remark that the Count-Min sketch [9] can be applied to approximate quantiles even in the presence of duplication (see [7]), i.e., is duplicant-resilient. However, the approximation guarantee achieved by Count-Min sketch for quantiles is only *additive* (i.e., $\pm\varepsilon n$), in contrast to the stronger notion of *multiplicative* approximation (i.e., $(1 \pm \varepsilon)$) we seek in this paper. To the best of our knowledge, there is no similar result concerning duplicate-resilient sketches for multiplicative approximation of quantiles or the linear regression problem, and existing techniques do not seem to be applicable for our purpose. Indeed one of the primary technical contributions of this paper is designing provisioning schemes that can effectively deal with overlapping hypotheticals for quantiles and linear regression.

Further related work. *Provisioning*, in the sense used in this paper, originated in [13] together with a proposal for how to perform it taking advantage of provenance tracking. Answering queries under hypothetical updates is studied in [17, 4] but the focus there is on using a specialized warehouse to avoid transactional costs. We refer the interested reader to [13] for more related work.

Estimating the number of distinct elements (corresponding to the count query) has been studied extensively in data streams [16, 2, 5, 28] and distributed functional monitoring [11, 35]. For estimating quantiles in the data stream or the distributed model, [31, 18, 22, 9, 23, 36] achieve an additive error of εn for an input of length n, and [24, 8] achieve a (stronger guarantee of) $(1 \pm \varepsilon)$-approximation. Sampling and sketching techniques for ℓ_2-regression problem have also been studied in [14, 32, 15, 6] for either speeding up the computation or in data streams (see [30, 34] for excellent surveys on this topic).

2 Problem Statement

Hypotheticals. Fix a relational schema Σ. Our goal is to provision queries on Σ-instances. A *hypothetical* w.r.t. Σ is a computable function h that maps every Σ-instance I to a sub-instance $h(I) \subseteq I$. Formalisms for specifying hypotheticals are of course of interest (e.g., apply a selection predicate to each table in I) but we do not discuss them here because the results in this paper do not depend on them.

Scenarios. We will consider analyses (scenario explorations) that start from a finite set H of hypotheticals. A *scenario* is a non-empty set of hypotheticals $S \subseteq H$. The result of applying a scenario $S = \{h_1, \ldots, h_s\}$ to an instance I is defined as a sub-instance $I|_S = h_1(I) \cup \cdots \cup h_s(I)$. In other words, under S, if any $h \in S$ is said to be turned on (similarly, any $h \in H \setminus S$ is turned off), each turned on hypothetical h will retain the tuples $h(I)$ from I.

▶ **Definition 4** (Provisioning). Given a query Q, to *provision* Q means to design a pair of algorithms: (i) a **compression** algorithm that takes as input an instance I and a set H of hypotheticals, and outputs a data structure Γ called a **sketch**, and (ii) an **extraction** algorithm that for any scenario $S \subseteq H$, outputs $Q(I|_S)$ using only Γ (without access to I).

To be more specific, we assume the compression algorithm takes as input an instance I, and k hypotheticals $h_1, \ldots, h_k$ along with the sub-instances $h_1(I), \ldots, h_k(I)$ that they define. A hypothetical will be referred to by an index from $\{1, \ldots, k\}$, and the extraction algorithm will be given scenarios in the form of sets of such indices. Hence, we will refer to a scenario $S \subseteq H$ where $S = \{h_{i_1}, \ldots, h_{i_s}\}$ by abusing the notation as $S = \{i_1, \ldots, i_s\}$.

We call such a pair of compression and extraction algorithms a *provisioning scheme*. The compression algorithm runs only once; the extraction algorithm runs repeatedly for all the scenarios that an analyst wishes to explore. We refer to the time that the compression algorithm requires as the *compression time*, and the time that extraction algorithm requires for each scenario as the *extraction time*.

The definition above is not useful by itself for positive results because it allows for trivial space-inefficient solutions. For example, the definition is satisfied when the sketch Γ is defined to be a copy of I itself or, as mentioned earlier, a scenario-indexed collection of all the answers. Obtaining the answer for each scenario is immediate for either case, but such a sketch can be prohibitively large as the number of tuples in I could be enormous, and the number of scenarios is exponential in $|H|$.

This discussion leads us to consider complexity bounds on the size of the sketches.

▶ **Definition 5** (Compact provisioning). A query Q can be *compactly* provisioned if there exists a provisioning scheme for Q that given any input instance I and any set of hypotheticals H, constructs a sketch of size $\text{poly}(|H|, \log |I|)$ bits.

We make the following important remark about the restrictions made in Definitions 4 and 5.

▶ Remark. At first glance, the requirement that the input instance I cannot be examined *at all* during extraction may seem artificial, and the same might be said about the size of the sketch depending polynomially on $\log n$ rather than a more relaxed requirement. However, we show that our lower bound results hold *even if* a portion of size $o(n)$ of the input instance can be examined during extraction *after* the scenario is revealed and *even if* the space dependence of the sketch is only restricted to be $o(n)$ (instead of depending only polynomially on $\log n$). In spite of this, the positive results we obtain all use sketches with space that depend polynomially only on $\log n$ and does *not* require examining the original database

during the extraction. These calibration results further justify our design choices for compact provisioning.

Even though the definition of compact provisioning does not impose any restriction on either the compression time or the extraction time, all our positive results in this paper are supported by (efficient) polynomial time algorithm. Note that this is *data-scenario complexity*: we assume the size of the query (and the schema) to be a constant but we consider dependence on the size of the instance and the number of hypotheticals. Our negative results (lower bounds on the sketch size), on the other hand, hold even when the compression and the extraction algorithms are computationally unbounded.

Exact vs. approximate provisioning. Definition 5 focused on exact answers for the queries. While this is appropriate for, e.g., relational algebra queries, as we shall see, for queries that compute numerical answers such as aggregates and analytics, having the flexibility of answering queries approximately is essential for any interesting positive result.

▶ **Definition 6** (ε-provisioning)**.** For any $0 < \varepsilon < 1$, a query Q can be *ε-provisioned* if there exists a provisioning scheme for Q, whereby for each scenario S, the extraction algorithm outputs a $(1 \pm \varepsilon)$ approximation of $Q(I|_S)$, where I is the input instance.

We say a query Q can be *compactly* ε-provisioned if Q can be ε-provisioned by a provisioning scheme that, given any input instance I and any set of hypotheticals H, creates a sketch of size $\text{poly}(|H|, \log|I|, 1/\varepsilon)$ bits.

We emphasize that throughout this paper, we only consider the approximation guarantees which are *relative* (multiplicative) as opposed to the weaker notion of additive approximations. The precise definition of relative approximation guarantee will be provided for each query individually. Moreover, as expected, randomization will be put to good use in ε-provisioning. We therefore extend the definition to cover the provisioning schemes that use both randomization and approximation.

▶ **Definition 7.** For any $\varepsilon, \delta > 0$, an *(ε, δ)-provisioning scheme* for a query Q is a provisioning scheme where both compression and extraction algorithms are allowed to be randomized and the output *for every* scenario S is an $(1 \pm \varepsilon)$-approximation of $Q(I|_S)$ with probability $1 - \delta$. Moreover, the compression time of the scheme is $\text{poly}(|I|, |H|, 1/\varepsilon, \log(1/\delta))$ and extraction time is $\text{poly}(|\Gamma|)$.

An (ε, δ)-provisioning scheme is called *compact* iff it constructs sketches of size only $\text{poly}(|H|, \log|I|, 1/\varepsilon, \log(1/\delta))$ bits.

Note that in many applications, the size of the database is a very large number, and hence the exponent in the $\text{poly}(|I|)$-dependence of the compression time might become an issue. Therefore, we further define *(ε, δ)-linear provisioning scheme,* where the dependence of the compression time on $|I|$ is *essentially linear*, i.e., $O(|I|) \cdot \text{poly}(|H|, \log(|I|), 1/\varepsilon, \log(1/\delta))$. All our positive results for queries with numerical answers will be stated in terms of compact (ε, δ)-linear provisioning schemes, which ensure efficiency in both running time and sketch size.

Complex queries. Our main target consists of practical queries that combine logical, grouping, and numerical components. In Section 5, we focus on *complex queries* defined by a logical (relational algebra or Datalog) query that returns a set of tuples, followed by a group-by operation (on set of grouping attributes) and further followed by numerical query that is applied to each sets of tuples resulting from the grouping. This class of queries already covers

many practical examples. We observe that the output of such a complex query is a set of p tuples where p is *the number of distinct values taken by the grouping attributes*. Therefore, the size of any provisioning sketch must also depend on p. We show (in Theorem 19) that a sketch for a query that involves grouping can be obtained as a collection of p sketches. Hence, if each of the p sketches is of compact size (as in Definitions 5 and 7) and the value p itself is bounded by $\text{poly}(|H|, \log |I|)$, then the overall sketch for the complex query is also of compact size. Note that p is typically small for the kind of grouping used in practical analysis queries (e.g., number of products, number of departments, number of locations, etc.). Intuitively, an analyst would have trouble making sense of an output with a large number of tuples.

Notation. Throughout the paper, we denote by k the number of hypotheticals, and by n the size $|I|$ of the input instance. For any integer $m > 0$, $[m]$ denotes the set $\{1, 2, \ldots, m\}$. The $\tilde{O}(\cdot)$ notation suppresses $\log\log(n)$, $\log\log(1/\delta)$, $\log(1/\varepsilon)$, and $\log(k)$ factors. All logarithms are in base two unless stated otherwise.

3 Coverage: A "Hard" Problem for Provisioning

To establish our lower bounds in this paper, we introduce a "hard" problem called Coverage. Though not defined in the exact formalism of provisioning, the Coverage problem can be solved by many provisioning schemes using proper "reductions" and hence a lower bound for the Coverage problem can be used to establish similar lower bounds for provisioning various queries.

Informally speaking, in the Coverage problem, we are given a collection of k subsets of a universe $[n]$ and the goal is to "compress" this collection in order to answer to the questions in which indices of some subsets in the collection are provided and we need to figure out whether these subsets cover the universe $[n]$ or not. We are interested in compressing schemes for this problem that when answering each question, in addition to the already computed summary of the collection, also have a limited access to the original instance (see Remark 2 after Definition 5 for motivation of this modification). The Coverage problem is defined formally as follows.

▶ **Problem 8** (Coverage). *Suppose we are given a collection $\mathcal{S} = \{S_1, S_2, \ldots S_k\}$ of the subsets of $[n]$. The goal in the* Coverage *problem is to find a* compressing scheme *for $\mathcal{S}$, defined formally as a triple of algorithms:*

- *A* ***compression algorithm*** *which given the collection $\mathcal{S}$ creates a* data structure *D.*
- *An* ***examination algorithm*** *which given a subset of $[k]$, a* question*, $Q = \{i_1, \ldots, i_s\}$ and the data structure D, computes a set $J \subseteq [n]$ of indices and lookup for each $j \in J$ and each S_i ($i \in [k]$), whether $j \in S_i$ or not. The output of the examination algorithm is a tuple $S^J := (S_1^J, \ldots, S_k^J)$, where $S_i^J = S_i \cap J$.*
- *An* ***extraction algorithm*** *which given a question $\{i_1, \ldots, i_s\}$, the data structure D, and the tuple S^J, outputs "Yes", if $S_{i_1} \cup \ldots \cup S_{i_s} = [n]$ and "No" otherwise.*

We refer to the size of D, denoted by $|D|$, as the storage requirement of the compression scheme and to the size of J, denoted by $|J|$, as the examination requirement of the scheme. The above algorithms can all be randomized; in that case, we require that for *each* question Q, the final answer (of the extraction algorithm) to be correct with a probability at least 0.99. Note that this choice of constant is arbitrary and is used only to simplify the analysis; indeed, one can always amplify the probability of success by repeating the scheme constant number of times and return the majority answer.

While Coverage is not stated in the exact formalism of provisioning, the analogy between this problem and provisioning schemes should be clear. In particular, for our lower bound proofs for provisioning schemes, we can alter the Definition 4 to add an examination algorithm and allow a similar access to the original database to the provisioning scheme.

We establish the following lower bound on storage and examination requirement of any compressing scheme for the Coverage problem. The proof of this Theorem is deferred to the full version of the paper [3] (see Theorem 3.1).

▶ **Theorem 9.** *Any compressing scheme for the* Coverage *problem that answers each question correctly with probability at least* 0.99*, either has storage requirement or examination requirement of* $\min(2^{\Omega(k)}, \Omega(n))$ *bits.*

Allowing access to the original input in Theorem 9 makes the lower bound very robust. However, due to this property, the lower bound does not seem to follow from standard communication complexity lower bounds and hence we use an information-theoretic approach to prove this theorem directly, which may be of independent interest. We remark that since our other lower bounds are typically proven using a reduction from the Coverage problem, the properties in Theorem 9 (i.e., allowing randomization and $o(n)$ access to the database *after* being given the scenario) also hold for them and we do not mention this explicitly.

4 Numerical Queries

In this section, we study provisioning of numerical queries, i.e., queries that output some (rational) number(s) given a set of tuples. In particular, we investigate aggregation queries including count, sum, average, and quantiles (therefore min, max, median, rank, and percentile), and as a first step towards provisioning database-supported machine learning, linear (ℓ_2) regression. We assume that the relevant attribute values are rational numbers of the form a/b where both a, b are integers in range $[-W, W]$ for some $W > 0$.

4.1 The Count, Sum, and Average Queries

In this section, we study provisioning of the count, sum, and average queries, formally defined as follows. The answer to the count query is the number of tuples in the input instance. For the other two queries, we assume a relational schema with a binary relation containing two attributes: an *identifier (key)* and a *weight*. We say that a tuple x is smaller than the tuple y, if the weight of x is smaller than the weight of y. Given an instance I, the answer to the sum query (resp. the average query) is the *total weights* of the tuples (resp. the *average weight* of the tuples) in I.

We first show that none of the count, sum, average queries can be provisioned both compactly and exactly, which motivates the ε-provisioning approach, and then briefly describe how to build a compact (ε, δ)-linear provisioning scheme for each of them.

▶ **Theorem 10.** *Exact provisioning of the* count*,* sum*, or* average *queries requires sketches of size* $\min(2^{\Omega(k)}, \Omega(n))$ *bits.*

Proof Sketch. We prove the lower bound for the count using a reduction from the Coverage problem; the lower bound of the sum follows immediately by setting all weights to be 1. The reduction for the average query is slightly more involved and is deferred to the full version of the paper [3] (see Theorem 4.1).

Given $\{S_1, \ldots, S_k\}$, where each S_i is a subset of $[n]$, we solve Coverage using a provisioning scheme for the count query. Define an instance I of a relational schema with a unary

relation A, where $I = \{A(x)\}_{x\in[n]}$. Define a set H of k hypotheticals, where for any $i \in [k]$, $h_i(I) = \{A(x)\}_{x\in S_i}$. For any scenario $S = \{i_1, \ldots, i_s\}$, the count of $I|_S$ is n iff $S_{i_1} \cup \ldots \cup S_{i_s} = [n]$. Hence, any provisioning scheme for the count query solves the Coverage problem and the lower bound follows from Theorem 9. ◀

We further point out that if the weights can be both positive and negative, the sum (and average) cannot be compactly provisioned even approximately (see the full version [3], Theorem 4.2), and hence we will focus on ε-provisioning for *positive* weights.

We conclude this section by briefly explaining the ε-provisioning schemes for the count, sum, and average queries. These results are mostly direct application of known techniques and we present them here for completeness.

▶ **Theorem 11** (count, sum, average)**.** *For any $\varepsilon, \delta > 0$, there exist compact (ε, δ)-linear provisioning schemes for the* count *query and the* sum*/*average *queries (with positive weights), respectively.*

The count query can be provisioned by using *linear sketches* for estimating the ℓ_0-norm (see, e.g., [28]) as follows. Consider each hypothetical $h_i(I)$ as an n-dimensional boolean vector $\mathbf{x}_i$, where the j-th entry is 1 iff the j-th tuple in I belongs to $h_i(I)$. For each $\mathbf{x}_i$, create a linear sketch (using $\tilde{O}(\varepsilon^{-2}\log n)$ bits of space) that estimates the ℓ_0-norm [28]. Given any scenario S, combine (i.e., add together) the linear sketches of the hypotheticals in S and use the combined sketch to estimate the ℓ_0-norm (which is equal to the answer of count).

Note that we can directly use linear sketching for provisioning the count query since counting the duplicates once (as done by union) or multiple times (as done by addition) does not change the answer. However, this is *not* the case for other queries of interest like quantiles and regression and hence linear sketching is not directly applicable for them.

In the full version of the paper [3] (see Theorem 4.3), we describe a self-contained approach for ε-provisioning the count query with a slightly better dependence on the parameter n ($\log\log n$ instead of $\log n$). We name this sketch the CNT-Sketch, which will be used used later for provisioning other queries. In particular, provisioning the sum query using a CNT-Sketch is straightforward when the weights are positive: group the tuples by weight into $\Theta(\log n/\varepsilon)$ groups and construct a CNT-Sketch for each group to estimate the total sum. In the full version [3] (see Theorem 4.4), we describe this in more detail and point out how to further improve the sketch size. The provisioning scheme for average follows immediately from these results.

4.2 The Quantiles Query

We now study provisioning of the quantiles query. We again assume a relational schema with just one binary relation containing attributes identifier and weight. For any instance I and any tuple $x \in I$, we define the *rank* of x to be the number of tuples in I that are smaller than or equal to x (in terms of the weights). The output of a quantiles query with a given parameter $\phi \in (0, 1]$ on an instance I is the tuple with rank $\lceil \phi \cdot |I| \rceil$. Finally, we say a tuple x is a $(1 \pm \varepsilon)$-approximation of a quantiles query whose correct answer is y, iff the rank of x is a $(1 \pm \varepsilon)$-approximation of the rank of y.

Similar to the previous section, we first show that the quantiles query admits no compact provisioning scheme for exact answer and then provide a compact ε-provisioning scheme for this query.

▶ **Theorem 12.** *Exact provisioning of the* quantiles *query* even on disjoint hypotheticals *requires sketches of size* $\min(2^{\Omega(k)}, \Omega(n))$ *bits.*

In the quantiles query, the parameter ϕ may be given either already to the compression algorithm or only to the extraction algorithm. The latter yields an immediate lower bound of $\Omega(n)$, since by varying ϕ over $(0, 1]$, one can effectively "reconstruct" the original database. However, we achieve a more interesting lower bound for the case when ϕ is given at to the compression algorithm (i.e., a fixed ϕ for all scenarios, e.g., setting $\phi = 1/2$ to find the *median*). An important property of the lower bound for quantiles is that, in contrast to all other lower bounds for numerical queries in this paper, this lower bound holds even for disjoint hypotheticals[2]. The proof of Theorem 12 is deferred to the full version [3] (see Theorem 4.7).

We now turn to establish the main result of this section, which argue the existence of a compact scheme for ε-provisioning the quantiles. We emphasize that the approximation guarantee in the following theorem is *multiplicative*.

▶ **Theorem 13** (quantiles). *For any $\varepsilon, \delta > 0$, there exists a compact (ε, δ)-linear provisioning scheme for the* quantiles *query that creates a sketch of size $\tilde{O}(k\varepsilon^{-3} \log n \cdot (\log(n/\delta)+k)(\log W + k))$ bits.*

We should note that in this theorem the parameter ϕ is only provided in the extraction phase. Our starting point is the following simple lemma first introduced by [24].

▶ **Lemma 14** ([24]). *For any list of* unique *numbers $A = (a_1, \ldots, a_n)$ and parameters $\varepsilon, \delta > 0$, let $t = \lceil 12\varepsilon^{-2} \log(1/\delta) \rceil$; for any* target rank *$r > t$, if we independently sample each element with probability t/r, then with probability at least $1 - \delta$, the rank of the t-th smallest sampled element is a $(1 \pm \varepsilon)$-approximation of r.*

The proof of Lemma 14 is an standard application of the Chernoff bound and the main challenge for provisioning the quantiles query comes from the fact that hypotheticals overlap. We propose the following scheme which addresses this challenge.

Compression algorithm for the quantiles **query.** Given an instance I, a set H of hypotheticals, and two parameters $\varepsilon, \delta > 0$, let $\varepsilon' = \varepsilon/5$, $\delta' = \delta/3$, and $t = \lceil 12\varepsilon'^{-2}(\log(1/\delta') + 2k + \log(n)) \rceil$.

1. Create and **record** a CNT-Sketch for I and H with parameters ε' and δ'.
2. Let $\{r_j = (1+\varepsilon')^j\}_{j=0}^{\lceil \log_{(1+\varepsilon')} n \rceil}$. For each r_j, create the following sub-sketch individually.
3. If $r_j \leq t$, for each hypothetical h_i, **record** the r_j smallest chosen tuples in $h_i(I)$. If $r_j > t$, for each hypothetical h_i, choose each tuple in $h_i(I)$ with probability t/r_j, and **record** the $\lceil (1 + 3\varepsilon') \cdot t \rceil$ smallest tuples in a list $T_{i,j}$. For each tuple x in the resulting list $T_{i,j}$, **record** its *characteristics vector* for the set of the hypotheticals, which is a k-dimensional binary vector $(v_1, v_2, \ldots, v_k)$, with value 1 on v_l whenever $x \in h_l(I)$ and 0 elsewhere.

[2] All other numerical queries that we study in this paper can be compactly provisioned for exact answer, when the hypotheticals are *disjoint*.

Extraction algorithm for the quantiles query. Suppose we are given a scenario S and a parameter $\phi \in (0, 1]$. In the following, the rank of a tuple always refers to its rank in the sub-instance $I|_S$.

1. Denote by $\tilde{n}$ the output of the CNT-Sketch on S. Let $\tilde{r} = \phi \cdot \tilde{n}$, and find the index γ, such that $r_\gamma \leq \tilde{r} < r_{\gamma+1}$.
2. If $r_\gamma \leq t$, among all the hypotheticals turned on by S, take the union of the recorded tuples and output the r_γ-th smallest tuple in the union.
3. If $r_\gamma > t$, from each h_i turned on by S, and each tuple x recorded in $T_{i,\gamma}$ with a characteristic vector $(v_1, v_2, \ldots, v_k)$, *collect* x iff for any $l < i$, either $v_l = 0$ or $h_l \notin S$. In other words, a tuple x recorded by h_i is taken only when among the hypotheticals that are turned on by S, i is the smallest index s.t. $x \in h_i(I)$. We will refer to this procedure as the *deduplication*. Output the t-th smallest tuple among all the tuples that are collected.

We call a sketch created by the above compression algorithm a QTL-Sketch. We defer the analysis of this sketch and the proof of Theorem 13 to the full version of the paper [3] (see Theorem 4.8).

Extensions. By simple extensions of our scheme, many variations of the quantiles query can be answered, including outputting the rank of a tuple x, the percentiles (the rank of x divided by the size of the input), or the tuple whose rank is Δ larger than x, where $\Delta > 0$ is a given parameter. As an example, for finding the rank of a tuple x, we can find the tuples with ranks approximately $\{(1+\varepsilon)^l\}$, $l \in [\lceil \log_{(1+\varepsilon)} n \rceil]$, using the QTL-Sketch, and among the found tuples, output the rank of the tuple whose weight is the closest to the weight of x.

4.3 The Linear Regression Query

In this section, we study provisioning of the regression query (i.e., the ℓ_2-regression problem), where the input is a matrix $\mathbf{A}_{n\times d}$ and a vector $\mathbf{b}_{n\times 1}$, and the goal is to output a vector $\mathbf{x}$ that minimizes $\|\mathbf{Ax} - \mathbf{b}\|$ ($\|\cdot\|$ stands for the ℓ_2 norm). A $(1+\varepsilon)$-approximation of the regression query is a vector $\tilde{\mathbf{x}}$ such that $\|\mathbf{A}\tilde{\mathbf{x}} - \mathbf{b}\|$ is at most $(1+\varepsilon)\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|$.

The input is specified using a relational schema Σ with a $(d+2)$-ary relation R. Given an instance I of Σ with n tuples, we interpret the projection of R onto its first d columns, the $(d+1)$-th column, and the $(d+2)$-column respectively as the matrix $\mathbf{A}$, the column vector $\mathbf{b}$, and the identifiers for the tuples in R. For simplicity, we denote $I = (\mathbf{A}, \mathbf{b})$, assume that the tuples are ordered, and use the terms the i-th tuple of I and the i-th row of $(\mathbf{A}, \mathbf{b})$ interchangeably.

Notation. For any matrix $\mathbf{M} \in \mathbb{R}^{n\times d}$, denote by $\mathbf{M}_{(i)}$ the i-th row of $\mathbf{M}$, and by $\mathbf{U}_M \in \mathbb{R}^{n\times\rho}$ (where ρ is the rank of $\mathbf{M}$) the orthonormal matrix of the column space of $\mathbf{M}$ (see [26] for more details). Given an instance $I = (\mathbf{A}, \mathbf{b})$, and k hypotheticals, we denote for each hypothetical h_i the sub-instance $h_i(I) = (\mathbf{A}_i, \mathbf{b}_i)$. For any integer i, $\mathbf{e}_i$ denotes the i-th standard basis; hence, the i-th row of $\mathbf{M}$ can be written as $\mathbf{e}_i^T\mathbf{M}$.

The following theorem shows that the regression query cannot be compactly provisioned for exact answers (see the full version [3], Theorem 4.10) and hence, we will focus on ε-provisioning.

▶ **Theorem 15.** *Exact provisioning of the* regression *query,* even when the dimension is $d = 1$, *requires sketches of size* $\min(2^{\Omega(k)}, \Omega(n))$ *bits*

Before continuing, we remark that if hypotheticals are disjoint, the regression query admits compact provisioning for exact answer (see the full version [3], Section 4.3), and hence the hardness of the problem again lies on the fact that hypotheticals overlap. We now turn to provide a provisioning scheme for the regression query and prove the following theorem.

▶ **Theorem 16** (regression)**.** *For any* $\varepsilon, \delta > 0$*, there exists a compact* (ε, δ)*-linear provisioning scheme for the* regression *query that creates a sketch of size* $\tilde{O}(\varepsilon^{-1}k^3 d \log(nW)(k + \log \frac{1}{\delta}))$ *bits.*

Overview. Our starting point is a non-uniform sampling based approach (originally used for speeding up the ℓ_2-regression computation [32]) which uses a small sample to accurately approximates the ℓ_2-regression problem. Since the probability of sampling a tuple (i.e., a row of the input) in this approach depends on its relative importance which can vary dramatically when input changes, this approach is not directly applicable to our setting.

Our contribution is a *two-phase sampling* based approach to achieve the desired sampling probability distribution for *any* scenario. At a high level, we first sample and record a small number of tuples from each hypothetical using the non-uniform sampling approach; then, given the scenario in the extraction phase, we re-sample from the recorded tuples of the hypotheticals presented in the scenario. Furthermore, to rescale the sampled tuples (as needed in the original approach), we obtain the exact sampling probabilities of the recorded tuples by recording their relative importance in each hypothetical. Our approach relies on a monotonicity property of the relative importance of a tuple when new tuples are added to the original input.

RowSample. We start by describing the non-uniform sampling algorithm. Let $\mathcal{P} = (p_1, p_2, \ldots, p_n)$ be a probability distribution, and $r > 0$ be an integer. Sample r tuples of $I = (\mathbf{A}, \mathbf{b})$ with replacement according to the probability distribution $\mathcal{P}$. For each sample, if the j-th row of $\mathbf{A}$ is sampled for some j, rescale the row with a factor $(1/\sqrt{rp_j})$ and store it in the sampling matrix $(\tilde{\mathbf{A}}, \tilde{\mathbf{b}})$. In other words, if the i-th sample is the j-th row of I, then $(\tilde{\mathbf{A}}_{(i)}, \tilde{\mathbf{b}}_{(i)}) = (\mathbf{A}_{(j)}, \mathbf{b}_{(j)})/\sqrt{rp_j}$. We denote this procedure by RowSample$(\mathbf{A}, \mathbf{b}, \mathcal{P}, r)$, and $(\tilde{\mathbf{A}}, \tilde{\mathbf{b}})$ is its output. The RowSample procedure has the following property [32] (see also [14, 34] for more details on introducing the parameter β).

▶ **Lemma 17** ([32])**.** *Suppose* $\boldsymbol{A} \in \mathbb{R}^{n \times d}$*,* $\boldsymbol{b} \in \mathbb{R}^n$*, and* $\beta \in (0, 1]$*;* $\mathcal{P} = (p_1, p_2, \ldots, p_n)$ *is a probability distribution on* $[n]$*, and* $r > 0$ *is an integer. Let* $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}})$ *be an output of* RowSample$(\boldsymbol{A}, \boldsymbol{b}, \mathcal{P}, r)$*, and* $\tilde{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\tilde{\boldsymbol{A}}\boldsymbol{x} - \tilde{\boldsymbol{b}}\|$.

If for all $i \in [n]$*,* $p_i \geq \beta \frac{\|e_i^T \boldsymbol{U}_A\|^2}{\sum_{j=1}^n \|e_j^T \boldsymbol{U}_A\|^2}$ *, and* $r = \Theta(\frac{d \log d \log(1/\delta)}{\varepsilon \cdot \beta})$*, then with probability at least* $(1 - \delta)$*,* $\|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b}\| \leq (1 + \varepsilon) \min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|$.

The value $\|e_i^T \mathbf{U}_A\|^2$, i.e the square norm of the i-th row of $\mathbf{U}_A$, is also called the *leverage score* of the i-th row of $\mathbf{A}$. One should view the leverage scores as the "relative importance" of a row for the ℓ_2-regression problem (see [30] for more details). Moreover, using the fact that columns of $\mathbf{U}_A$ are orthonormal, we have $\sum_j \|e_j^T \mathbf{U}_A\|^2 = \rho$, where ρ is the rank of $\mathbf{A}$.

We now define our provisioning scheme for the regression query, where the compression algorithm performs the first phase of sampling (samples rows from each hypothetical) and the extraction algorithm performs the second (samples from the recorded tuples).

Compression algorithm for the regression query. Suppose we are given an instance $I = (\mathbf{A}, \mathbf{b})$ and k hypotheticals with $h_i(I) = (\mathbf{A}_i, \mathbf{b}_i)$ $(i \in [k])$. Let $t = \Theta(\varepsilon^{-1} kd \log d \cdot (k + \log(1/\delta)))$, and define for each $i \in [k]$ a probability distribution $\mathcal{P}_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,n})$ as follows. If the j-th row of $\mathbf{A}$ is the l-th row of $\mathbf{A}_i$ (they correspond to the same tuple), let $p_{i,j} = \|e_l^T \mathbf{U}_{A_i}\|^2/\rho$, where ρ is the rank of $\mathbf{A}_i$. If $\mathbf{A}_{(j)}$ does not belong to $\mathbf{A}_i$, let $p_{i,j} = 0$. Using the fact that for every $i \in [k]$, $\mathbf{U}_{A_i}$ is an orthonormal matrix, $\sum_{j=1}^{n} p_{i,j} = 1$. **Record** t independently chosen random permutations of $[k]$, and for each hypothetical h_i, create a sub-sketch as follows.

1. Sample t tuples of $h_i(I)$ with replacement, according to the probability distribution $\mathcal{P}_i$.
2. For each of the sampled tuples, assuming it is the j-th tuple of I, **record** the tuple along with its sampling rate in each hypothetical, i.e., $\{p_{i',j}\}_{i' \in [k]}$.

Extraction algorithm for the regression query. Given a scenario $S = \{i_1, \ldots, i_s\}$, we will recover from the sketch a matrix $\tilde{\mathbf{A}}_{t \times d}$ and a vector $\tilde{\mathbf{b}}_{t \times 1}$. For $l = 1$ to t:

1. Pick the l-th random permutation recorded in the sketch. Let γ be the first value in this permutation that appears in S.
2. If $(\mathbf{a}, b)$ is the l-th tuple sampled by the hypothetical h_γ, which is the j-th tuple of I, let $q_j = \sum_{i \in S} p_{i,j}/|S|$, using the recorded sampling rates.
3. Let $(\tilde{\mathbf{A}}_{(l)}, \tilde{\mathbf{b}}_{(l)}) = (\mathbf{a}, b)/\sqrt{tq_j}$. Return $\tilde{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\tilde{\mathbf{A}}\mathbf{x} - \tilde{\mathbf{b}}\|$ (using any standard method for solving the ℓ_2-regression problem).

We call a sketch constructed above a REG-Sketch. In order to show the correctness of this scheme, we need the following lemma regarding the monotonicity of leverage scores (the proof is presented in the full version [3], Lemma 4.13).

▶ **Lemma 18** (Monotonicity of Leverage Scores). *Let $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{B} \in \mathbb{R}^{m \times d}$ be any matrix. Define matrix $\boldsymbol{C} \in \mathbb{R}^{(n+m) \times d}$ by* appending *rows of $\boldsymbol{B}$ to $\boldsymbol{A}$, i.e., $\boldsymbol{C}^T = [\boldsymbol{A}^T, \boldsymbol{B}^T]$. For any $i \in [n]$, if L_i is the leverage score of $\boldsymbol{A}_{(i)}$ and $\hat{L}_i$ is the leverage score of $\boldsymbol{C}_{(i)}$, then $L_i \geq \hat{L}_i$.*

Lemma 18 claims that adding more rows to a matrix $\mathbf{A}$ can only reduce the importance of any point originally in $\mathbf{A}$. Note that this is true even when the matrix $\mathbf{C}$ is formed by arbitrarily combining rows of $\mathbf{B}$ and $\mathbf{A}$ (rather than appending at the end).

Proof of Theorem 16. Fix a scenario S and let $I|_S = (\hat{\mathbf{A}}, \hat{\mathbf{b}})$. It is straightforward to verify that, for any step $l \in [t]$, q_j (in line (2) of the extraction algorithm) is the probability that the j-th tuple of I is chosen, if the j-th tuple belongs to $I|_S$. Hence, assuming $\mathcal{P}'$ is the probability distribution defined by the q_js on rows of the $I|_S$, the extraction algorithm implements RowSample$(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \mathcal{P}', t)$.

We will show that $q_j \geq \|e_l^T \mathbf{U}_{\hat{A}}\|^2/k\hat{\rho}$, where the l-th row of $\hat{\mathbf{A}}$ is the j-th row of $\mathbf{A}$, and $\hat{\rho}$ is the rank of $\hat{A}$. Then, by Lemma 17 with β set to $1/k$, with probability at least $1 - \frac{\delta}{2^k}$, $\|\hat{\mathbf{A}}\tilde{\mathbf{x}} - \hat{\mathbf{b}}\|$ is at most $(1+\varepsilon)\min_{\mathbf{x}} \|\hat{\mathbf{A}}\mathbf{x} - \hat{\mathbf{b}}\|$; hence, the returned vector $\tilde{\mathbf{x}}$ is a $(1+\varepsilon)$-approximation. Applying a union bound over all 2^k scenarios, with probability at least $(1-\delta)$, our scheme ε-provisions the regression query.

We now prove that $q_j \geq \|e_l^T \mathbf{U}_{\hat{A}}\|^2/k\hat{\rho}$. Denote by $L_{i,j}$ (resp. $L_{S,j}$) the leverage score of the j-th tuple of I in the matrix $\mathbf{A}_i$ (resp. $\hat{\mathbf{A}}$). Further, denote by ρ_i the rank of $\mathbf{A}_i$. Consequently, $p_{i,j} = L_{i,j}/\rho_i$, and our goal is to show $q_j \geq L_{S,j}/(k\hat{\rho})$. Pick any $i^* \in S$ where

$h_{i^*}(I)$ contains the j-th tuple of I, then:

$$q_j = \frac{\sum_{i \in S} p_{i,j}}{s} \geq \frac{p_{i^*,j}}{s} \geq \frac{L_{i^*,j}}{k\rho_i} \geq \frac{L_{S,j}}{k\hat{\rho}} \tag{1}$$

For the last inequality, since $\mathbf{A}_i$ is a sub-matrix of $\hat{\mathbf{A}}$, $\rho_i \leq \hat{\rho}$ and the leverage score decreases due to the monotonicity (Lemma 18).

To conclude, the probabilities will be stored with precision $1/n$ (hence stored using $O(\log n)$ bits each), and the size of the sketch is straightforward to verify. ◀

5 Complex Queries

We study the provisioning of queries that combine logical components (relational algebra and Datalog), with grouping and with the numerical queries that we studied in Section 4.

We start by defining a class of such queries and their semantics formally. For the purposes of this paper, a *complex query* is a triple $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ where Q_L is a relational algebra or Datalog query that outputs some relation with attributes $\bar{A}\bar{B}$ for some $\bar{B}$, $G_{\bar{A}}$ is a *group-by* operation applied on the attributes $\bar{A}$, and Q_N is a numerical query that takes as input a relation with attributes $\bar{B}$. For any input I let $P = \Pi_{\bar{A}}(Q_L(I))$ be the $\bar{A}$-relation consisting of all the distinct values of the grouping attributes. For each tuple $\bar{u} \in P$, we define $\Gamma_{\bar{u}} = \{\bar{v} \mid \bar{u}\bar{v} \in Q_L(I)\}$. Then, the output of the complex query $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ is a set of tuples $\{\langle \bar{u}, Q_N(\Gamma_{\bar{u}}) \rangle \mid \bar{u} \in P\}$.

In the following, we give positive results for the case where the logical component is a *positive relational algebra* (i.e., SPJU) query. It will be convenient to assume a different, but equivalent, formalism for these logical queries, namely that of *unions of conjunctive queries* (UCQs)[3]. We review quickly the definition of UCQs. A *conjunctive query* (CQ) over a relational schema Σ is of the form $ans(\overline{x}) \ :- \ R_1(\overline{x}_1), \ldots, R_b(\overline{x}_b)$, where atoms $R_1, \ldots, R_b \in \Sigma$, and the *size* of a CQ is defined to be the number of atoms in its body (i.e., b). A union of conjunctive query (UCQ) is a finite union of some CQs in which the head has the same schema.

In the following theorem, we show that for any complex query, where the logical component is a positive relational algebra query, compact provisioning of the numerical component implies compact provisioning of the complex query itself.

▶ **Theorem 19.** *For any complex query $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ where Q_L is a* UCQ*, if the numerical component Q_N can be compactly provisioned (resp. compactly ε-provisioned), and if the number of groups is bounded by* $\mathrm{poly}(k, \log n)$*, then the query $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ can also be compactly provisioned (resp. compactly ε-provisioned with the same parameter ε).*

Proof. Suppose Q_N can be compactly provisioned (the following proof also works when Q_N can be compactly ε-provisioned). Let b be the maximum size of the conjunctive queries in Q_L. Given an input instance I and a set H of k hypotheticals, we define a new instance $\hat{I} = Q_L(I)$ and a set $\hat{H}$ of $O(k^b)$ new hypotheticals as follows. For each subset $S \subseteq [k]$ of size at most b (i.e., $|S| \leq b$), define a hypothetical $\hat{h}_S(\hat{I}) = Q_L(I|_S)$ (though S is not a number, we still use it as an index to refer to the hypothetical $\hat{h}_S$). By our definition of the semantics

[3] Although the translation of an SPJU query to a UCQ may incur an exponential size blowup [1], in this paper, query (and schema) size are assumed to be constant. In fact, in practice, SQL queries often present with unions already at top level.

of complex queries, the group-by operation partitions $\hat{I}$ and each $\hat{h}_S$ into $p = |\Pi_{\bar{A}}(\hat{I})|$ groups. We treat each group individually, and create a sketch for each of them.

To simplify the notation, we still use $\hat{I}$ and $\hat{H}$ to denote respectively the portion of the new instance, and the portion of each new hypothetical that correspond to, without loss of generality, the first group. In the following, we show that a compact provisioning scheme for Q_N with input $\hat{I}$ and $\hat{H}$ can be adapted to compactly provision $\langle Q_L; G_{\bar{A}}; Q_N \rangle$ for the first group. Since the number of groups p is assumed to be $\text{poly}(\log |I|, |H|)$, the overall sketch size is still $\text{poly}(\log |I|, |H|)$, hence achieving compact provisioning for the complex query.

Create a sketch for Q_N with input $\hat{I}$ and $\hat{H}$. For any scenario $S \in [k]$ (over H), we can answer the numerical query Q_N using the scenario $\hat{S}$ (over $\hat{H}$) where $\hat{S} = \{S' \mid S' \subseteq S \text{ and } |S'| \leq b\}$. To see this, we only need to show that the input to Q_N remains the same, i.e., $Q_L(I|_S)$ is equal to $\hat{I}|_{\hat{S}}$. Each tuple t in $Q_L(I|_S)$ can be derived using (at most) b hypotheticals. Since any subset of S with at most b hypotheticals belongs to $\hat{S}$, the tuple t belongs to $\hat{I}|_{\hat{S}}$. On the other hand, each tuple t' in $\hat{I}|_{\hat{S}}$ belongs to some $\hat{h}_{S'}$ where $S' \in S$, and hence, by definition of $\hat{h}_{S'}$, the tuple t is also in $Q_L(I|_S)$. Hence, $Q_L(I|_S) = \hat{I}|_{\hat{S}}$.

Consequently, any compact provisioning scheme for Q_N can be adapted to a compact provisioning scheme for the query $\langle Q_L; G_{\bar{A}}; Q_N \rangle$. ◀

Theorem 19 further motivates our results in Section 4 for numerical queries as they can be extended to these quite practical queries. Additionally, as an immediate corollary of the proof of Theorem 19, we obtain that any boolean UCQ (i.e., any UCQ that outputs a boolean answer rather than a set of tuples) can be compactly provisioned.

▶ **Corollary 20.** *Any boolean UCQ can be compactly provisioned using sketches of size $O(k^b)$ bits, where b is the maximum size of each CQ.*

▶ Remark. [13] introduced query provisioning from a practical perspective and proposed *boolean provenance* [27, 20, 19, 33] as a way of building sketches. This technique can also be used for compactly provisioning boolean UCQs (see the full version [3], Remark 5.3).

We further point out that the exponential dependence of the sketch size on the query size (implicit) in Theorem 19 and Corollary 20 cannot be avoided even for CQs (the proof is in the full version [3]; see Theorem 5.5).

▶ **Theorem 21.** *There exists a boolean conjunctive query Q of size b such that provisioning Q requires sketches of size $\min(\Omega(k^{b-1}), \Omega(n))$ bits.*

More general queries. It is natural to ask (a) if Theorem 19 still holds when adding *negation* or *recursion* to the query Q_L (i.e. UCQ *with negation* and *recursive Datalog*, respectively), and (b) whether or not it is possible to provision queries in which logical operations are done *after* numerical ones. A typical example of a query in part (b) is a selection on aggregate values specified by a HAVING clause. Unfortunately, the answer to both questions is negative.

We first show that the answer to question (a) is negative.

▶ **Theorem 22.** *Exact provisioning of (i) boolean conjunctive queries with negation, or (ii) recursive Datalog (even without negation) queries requires sketches of size $\min(2^{\Omega(k)}, \Omega(n))$.*

Proof Sketch. We sketch the proof of each part separately; the complete proofs can be found in the full version [3] (see Theorem 5.6).

Part (i). Define the following boolean conjunctive query with negation over a schema with two unary relation symbols, A and B:

$$Q_{\texttt{NOTSUB}}() \text{ :- } A(x), \neg B(x)$$

This query returns true on I iff there exists some x such that $A(x) \in I$ and $B(x) \notin I$. Intuitively, if we view A and B as subsets of the active domain of I, it is querying whether or not "B is a subset of A". We use a reduction from the Coverage problem to prove the lower bound for $Q_{\texttt{NOTSUB}}$.

From an instance $\{S_1, \ldots, S_k\}$ ($S_i \subseteq [n]$) of Coverage, we create the following instance I for the schema $\Sigma = \{A, B\}$, where for any $x \in [n]$, $A(x), B(x) \in I$. Define the set of of hypotheticals $H = \{h_1, h_2, \ldots, h_{k+1}\}$, where for any $i \in [k]$, $h_i(I) = \{B(x) \mid x \in S_i\}$ and $h_{k+1}(I) = \{A(x) \mid x \in [n]\}$. It is easy to see that for any set $\hat{S} = \{i_1, \ldots, i_s\} \subseteq [k]$, under the scenario $S = \hat{S} \cup \{k+1\}$, $Q_{\texttt{NOTSUB}}(I|_S)$ is true iff there exists $x \in [n]$ s.t. $B(x) \notin I|_S$ which is equivalent to $[n] \not\subseteq S_{i_1} \cup \ldots \cup S_{i_s}$. Therefore any provisioning scheme of $Q_{\texttt{NOTSUB}}$ solves the Coverage problem and the result follows from Theorem 9.

Part (ii). Consider the following Datalog query, st-connectivity:

$$ans() \text{ :- } T(\texttt{t}) \quad (2)$$
$$T(y) \text{ :- } E(x, y), T(x) \quad (3)$$
$$T(\texttt{s}) \text{ :- } \quad (4)$$

This query returns true iff there is a path from the vertex `s` to the vertex `t` in the digraph defined by E. To prove a lower bound for the st-connectivity query we use again a reduction from the Coverage problem.

From an instance $\{S_1, \ldots, S_k\}$ ($S_i \subseteq [n]$) of Coverage, we create the following graph $G(V, E)$ with vertex set $V = \{(s =) v_0, v_1, v_2, \ldots, v_n (= t)\}$, and edges $E(v_{j-1}, v_j)$, for all $j \in [n]$. In G, there is only one path from s to t and that path uses all the n edges. The edge set E of the graph G is the input I to the provisioning scheme. Define the hypotheticals $H = \{h_1, h_2, \ldots, h_k\}$ where $h_i(I) = \{E(v_{j-1}, v_j)\}_{j \in S_i}$, for all $i \in [k]$. For any scenario $S = \{i_1, \ldots, i_s\} \subseteq [k]$, $T(t)$ is true (i.e., s is connected to t) in $I|_S$ iff all n edges are in $I|_S$, which is equivalent to $S_{i_1} \cup \ldots \cup S_{i_s} = [n]$. Therefore any provisioning scheme of st-connectivity solves the Coverage problem and the result follows from Theorem 9. ◀

Showing a negative answer to question (b) is very easy. As we already showed in Theorems 10 and 12, there are numerical queries that do not admit compact provisioning for *exact* answer. One can simply verify that each of those queries can act as a counter example for question (b) by considering HAVING clauses that test the equality of the answer to the numerical part against an exact answer (e.g. testing whether the answer to count is n or not).

6 Conclusions and Future Work

In this paper, we initiated a formal framework to study compact provisioning schemes for relational algebra queries, statistics/analytics including quantiles and linear regression, and complex queries. We considered provisioning for exact as well as approximate answers, and established upper and lower bounds on the sizes of the provisioning sketches.

The queries in our study include quantiles and linear regression queries from the list of in-database analytics highlighted in [25]. This is only a first step and the study of provisioning

for other core analytics problems, such as variance computation, k-means clustering, logistic regression, and support vector machines is of interest.

Another direction for future research is the study of queries in which *numerical* computations follow each other (e.g., when the linear regression training data is itself the result of aggregations). Yet another direction for future research is an extension of our model to allow other kinds of hypotheticals/scenarios as discussed in [13] that are also of practical interest. For example, an alternative natural interpretation of hypotheticals is that they represent tuples to be *deleted* rather than retained. Hence the application of a scenario $S \subseteq [k]$ to I becomes $I|_S = I \setminus (\bigcup_{i \in S} h_i(I))$. Using our lower bound techniques, one can easily show that even simple queries like count or sum cannot be approximated to within any multiplicative factor under this definition. Nevertheless, it will be interesting to identify query classes that admit compact provisioning in the delete model or alternative natural models.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29. ACM, 1996.
3. Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen. Algorithms for provisioning queries and analytics. *CoRR*, abs/1512.06143, 2015. URL: `http://arxiv.org/abs/1512.06143`.
4. Andrey Balmin, Thanos Papadimitriou, and Yannis Papakonstantinou. Hypothetical queries in an OLAP environment. In *VLDB*, pages 220–231, 2000. URL: `http://www.vldb.org/conf/2000/P220.pdf`.
5. Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM*. Springer, 2002.
6. Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *STOC*, pages 205–214. ACM, 2009.
7. G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *ICDE*, pages 20–31, 2006. `doi:10.1109/ICDE.2006.173`.
8. Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In *PODS*, pages 263–272. ACM, 2006.
9. Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1), 2005.
10. Graham Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS*, pages 271–282, 2005.
11. Graham Cormode, S Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. *ACM Transactions on Algorithms (TALG)*, 7(2):21, 2011.
12. Graham Cormode, S Muthukrishnan, Ke Yi, and Qin Zhang. Optimal sampling from distributed streams. In *PODS*, pages 77–86. ACM, 2010.
13. Daniel Deutch, Zachary G Ives, Tova Milo, and Val Tannen. Caravan: Provisioning for what-if analysis. In *CIDR*, 2013.
14. Petros Drineas, Michael W Mahoney, and S Muthukrishnan. Sampling algorithms for ℓ_2 regression and applications. In *SODA*, pages 1127–1136. ACM, 2006.
15. Petros Drineas, Michael W Mahoney, S Muthukrishnan, and Tamás Sarlós. Faster least squares approximation. *Numerische Mathematik*, 117(2):219–249, 2011.
16. Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

17 Shahram Ghandeharizadeh, Richard Hull, and Dean Jacobs. Heraclitus: Elevating deltas to be first-class citizens in a database programming language. *ACM Trans. Database Syst.*, 21(3):370–426, 1996. doi:10.1145/232753.232801.

18 Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin J Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *PVLDB*, pages 454–465. VLDB Endowment, 2002.

19 T.J. Green. Containment of conjunctive queries on annotated relations. *Theory Comput. Syst.*, 49(2), 2011.

20 T.J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

21 Greenplum DB (Pivotal). URL: http://pivotal.io/big-data/pivotal-greenplum-database.

22 Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.

23 Michael B Greenwald and Sanjeev Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS*, pages 275–285. ACM, 2004.

24 Anupam Gupta and Francis X Zane. Counting inversions in lists. In *SODA*, pages 253–254. Society for Industrial and Applied Mathematics, 2003.

25 Joseph M. Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. The madlib analytics library or MAD skills, the SQL. *PVLDB*, 5(12):1700–1711, 2012. URL: http://vldb.org/pvldb/vol5/p1700_joehellerstein_vldb2012.pdf.

26 Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

27 T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

28 Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52. ACM, 2010.

29 The MADlib Project. URL: http://madlib.net.

30 Michael W Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

31 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record*, 27(2):426–435, 1998.

32 Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *FOCS*, pages 143–152. IEEE, 2006.

33 D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

34 David P Woodruff. Sketching as a tool for numerical linear algebra. *arXiv:1411.4357*, 2014.

35 David P Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *STOC*, pages 941–960. ACM, 2012.

36 Ke Yi and Qin Zhang. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica*, 65(1):206–223, 2013.

Limits of Schema Mappings

Phokion G. Kolaitis[1], Reinhard Pichler[2], Emanuel Sallinger[3], and Vadim Savenkov[4]

1 **University of California – Santa Cruz, Santa Cruz, USA; and IBM Research-Almaden, San Jose, USA**
2 **TU Wien, Vienna, Austria**
3 **University of Oxford, Oxford, UK**
4 **Vienna University of Economics and Business, Vienna, Austria**

Abstract

Schema mappings have been extensively studied in the context of data exchange and data integration, where they have turned out to be the right level of abstraction for formalizing data inter-operability tasks. Up to now and for the most part, schema mappings have been studied as static objects, in the sense that each time the focus has been on a single schema mapping of interest or, in the case of composition, on a pair of schema mappings of interest.

In this paper, we adopt a dynamic viewpoint and embark on a study of sequences of schema mappings and of the limiting behavior of such sequences. To this effect, we first introduce a natural notion of distance on sets of finite target instances that expresses how "close" two sets of target instances are as regards the certain answers of conjunctive queries on these sets. Using this notion of distance, we investigate pointwise limits and uniform limits of sequences of schema mappings, as well as the companion notions of pointwise Cauchy and uniformly Cauchy sequences of schema mappings. We obtain a number of results about the limits of sequences of GAV schema mappings and the limits of sequences of LAV schema mappings that reveal striking differences between these two classes of schema mappings. We also consider the completion of the metric space of sets of target instances and obtain concrete representations of limits of sequences of schema mappings in terms of generalized schema mappings, i.e., schema mappings with infinite target instances as solutions to (finite) source instances.

1998 ACM Subject Classification H.2.0 Database Management – General

Keywords and phrases Limit, Pointwise convergence, Uniform convergence, Schema mapping

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.19

1 Introduction

Schema mappings have been extensively studied in the context of data exchange and data integration, where they have turned out to be the right level of abstraction for formalizing data inter-operability tasks (see the surveys [11, 12] and the monograph [1]). Up to now and for the most part, schema mappings have been studied as static objects, in the sense that each time the focus has been on a single schema mapping or on a finite and, typically, small number of schema mappings. In the case of data exchange [6], a single schema mapping is used to specify the relationship between a source schema and a target schema. In the case of operators on schema mappings [3], such as the composition operator [14, 8], a fixed number of schema mappings is used as input (e.g., two schema mappings in the case of composition) and return another schema mapping as output. Even the case of schema-mapping evolution [9] entails a finite (but potentially large) number of schema mappings.

In this paper, we adopt a dynamic viewpoint and embark on a systematic investigation of sequences of schema mappings and of the limiting behavior of such sequences. The original motivation came from the earlier work [2, 5, 7, 10, 14] on schema-mapping optimization and the study of various notions of equivalence between schema mappings that, intuitively, stipulate that two schema mappings cannot be distinguished using conjunctive queries (CQ-equivalence) or conjunctive queries with at most n variables (CQ_n-equivalence), for some fixed $n \geq 1$. In particular, in [5] and, implicitly, in [14], it was shown that, given an SO-tgd (second-order tuple-generating dependency) σ and a positive integer n, one can construct a GLAV schema mapping that is CQ_n-equivalent to σ. Informally, this means that a given SO tgd can be "approximated" by GLAV schema mappings up to any fixed level of precision, even though an SO tgd is a formula of second-order logic that may not be logically equivalent to any formula of first-order logic and, in particular, to any GLAV schema mapping. A more dynamic interpretation is that, given an SO-tgd σ, one can obtain a sequence of GLAV schema mappings $(\mathcal{M}_n)_{n\geq 1}$, whose "limit" is σ.

Summary of Results. Our contributions are both conceptual and technical. At the conceptual level, we develop a framework for studying sequences of schema mappings by first introducing a natural notion of distance dist on the powerset $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ of the set $\mathrm{Inst}(\mathbf{T})$ of finite instances over a schema $\mathbf{T}$. Intuitively, this notion of distance expresses how "close" two sets of finite $\mathbf{T}$-instances are as regards the certain answers of conjunctive queries on these sets. The pair $(\mathcal{P}(\mathrm{Inst}(\mathbf{T})), \mathrm{dist})$ is a pseudometric space, which means that the distance function dist is symmetric and obeys the triangle inequality, but different sets of finite target instances may have distance zero; however, two such sets have distance zero if and only if they are CQ-equivalent, i.e., every conjunctive query has the same certain answers on these two sets. Thus, we will also work with the metric space obtained by considering the CQ-equivalence classes of members of $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$, and will use the same notation for it.

Sequences of functions from some set to a metric space occupy a central place in the study of metric spaces (see, e.g., [18]). In particular, there are natural notions of a *pointwise limit* and of a *uniform limit* of a sequence $(f_n)_{n\geq 1}$ of functions from some set to a metric space; moreover, there are companion notions of a *pointwise Cauchy* and of a *uniformly Cauchy* sequence of such functions. We now describe briefly how these notions can be applied to sequences of schema mappings. In its most general formulation, a schema mapping $\mathcal{M}$ over a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$ is a set of pairs (I, J), where I is a finite $\mathbf{S}$-instance and J is a finite $\mathbf{T}$-instance. It follows that a schema mapping $\mathcal{M}$ can be also be viewed as a function f from the set $\mathrm{Inst}(\mathbf{S})$ of all finite $\mathbf{S}$-instances to the powerset $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ of the set of all finite $\mathbf{T}$-instances, where $f(I) = \{J : (I, J) \in \mathcal{M}\}$. This way, a sequence $(\mathcal{M}_n)_{n\geq 1}$ of schema mappings over a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$ can be viewed as a sequence of functions from $\mathrm{Inst}(\mathbf{S})$ to the (pseudo)metric space $(\mathcal{P}(\mathrm{Inst}(\mathbf{T})), \mathrm{dist})$.

After the conceptual framework has been laid out, we study in depth the limiting behavior of sequences of GAV mappings and the convergence of sequences of LAV mappings. We establish a number of technical results that reveal rather dramatic and perhaps unanticipated differences between GAV schema mappings and LAV schema mappings.

For sequences of GAV mappings, we point out that every uniformly Cauchy sequence of GAV mappings is eventually constant, hence it has a GAV mapping as uniform limit. We also show that every pointwise Cauchy sequence of GAV mappings has a pointwise limit, but it need not have a uniform limit; moreover, there are pointwise Cauchy sequences of GAV mappings such that no GAV mapping is their pointwise limit. This raises the question as to when a sequence of GAV mapping has a GAV mapping as a pointwise limit. We prove that

a sequence of GAV mappings has a GAV mapping as a pointwise limit if and only if it has a pointwise limit that allows for CQ-rewriting[1].

For sequences of LAV mappings, we show that the notions of uniform limit and pointwise limit coincide; moreover, the same holds true for the notions of uniformly Cauchy and pointwise Cauchy sequences. However, there are uniformly Cauchy sequences of LAV mappings that have no uniform limit. We also establish that a uniformly Cauchy sequence of LAV mappings has a LAV mapping as a uniform limit if and only if it has a uniform limit that admits universal solutions. The aforementioned results lift to sequences of *premise-bounded* sequences of GLAV mappings, i.e., sequences of GLAV mappings for which there is a $k \geq 1$ such that, for every mapping in the sequence, the left-hand side of every GLAV constraint has at most k source atoms (LAV mappings have $k = 1$).

In terms of techniques, we use systematically the structural characterizations of schema-mapping languages established in [19], thus creating a link with a different line of research.

The metric space $(\mathcal{P}(\text{Inst}(\mathbf{T})), \text{dist})$ is incomplete, i.e., there are Cauchy sequences of elements of $\mathcal{P}(\text{Inst}(\mathbf{T}))$ that have no limit in $\mathcal{P}(\text{Inst}(\mathbf{T}))$. It is well known that every incomplete metric space (X, d) has a completion, which means that it can be embedded into a complete metric space (X^*, d^*) so that X is a dense subset of X^*. Moreover, pointwise (respectively, uniformly) Cauchy sequences of functions on X have pointwise (respectively, uniform) limits that take values in X^*. The construction of X^* from X involves equivalence classes of Cauchy sequences of elements of X, thus, in general, the members of X^* do not have a concrete representation. In the last part of the paper, we show that the members of $\mathcal{P}(\text{Inst}(\mathbf{T}))^*$ can be represented by suitably constructed infinite $\mathbf{T}$-instances. As a consequence of this, the pointwise (respectively, uniform) limits of Cauchy sequences of schema mappings can be represented by *generalized* schema mappings, i.e., schema mappings that allow for infinite target instances as solutions to finite source instances.

2 Preliminaries

This section contains a minimum amount of the necessary background material.

Schemas, Instances, and Conjunctive Queries. A *schema* $\mathbf{R}$ is a finite sequence $\langle R_1, \ldots, R_k \rangle$ of relation symbols, where each R_i has a fixed arity. An *instance* I over $\mathbf{R}$, or an $\mathbf{R}$-*instance*, is a sequence $(R_1^I, \ldots, R_k^I)$, where each R_i^I is a finite relation of the same arity as R_i. We will often use R_i to denote both the relation symbol and the relation R_i^I that interprets it. The *active domain* of an instance I is the set of all values occurring in the relations of I. A *fact* of an instance I (over $\mathbf{R}$) is an expression $R_i^I(a_1, \ldots, a_m)$ (or simply $R_i(v_1, \ldots, v_m)$), where R_i is a relation symbol of $\mathbf{R}$ and $(a_1, \ldots, a_m) \in R_i^I$.

A *conjunctive query* is a first-order formula of the form $\exists \mathbf{z}\, \theta(\mathbf{x}, \mathbf{z})$, where $\theta(\mathbf{x}, \mathbf{z})$ is a conjunction of atomic formulas $R_i(v_1, ..., v_m)$ and each v_j is one of the variables in $\mathbf{x}$ and $\mathbf{z}$. A *boolean conjunctive query* is a conjunctive query with no free variables. We write CQ for the class of all conjunctive queries over some schema. For every $n \geq 1$, we let CQ_n denote the class of all conjunctive queries with at most n variables. We also let CQ_0 denote the singleton consisting of a trivially true query.

Schema Mappings, Universal Solutions, Certain Answers. Motivated by the terminology in data exchange [6], we typically work with two schemas, a *source schema* $\mathbf{S}$ and a *target*

[1] Allowing for CQ-rewriting means that the certain answers of every conjunctive query over the target schema is definable by a union of conjunctive queries over the source schema - see [19].

schema $\mathbf{T}$ with no relation symbols in common. We refer to $\mathbf{S}$-instances as *source instances*, and to $\mathbf{T}$-instances as *target instances*. We assume the presence of two kinds of values in instances, namely *constants* and *(labeled) nulls*. We also assume that the active domains of source instances consists of constants; the active domains of target instances may contain both constants and nulls.

A *schema mapping* $\mathcal{M}$ between a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$ is a set of pairs (I, J), where I is source instance and J a target instance. A schema mapping is often (but not always) given as a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a set of formulas in some suitable logical formalism such that $(I, J) \in \mathcal{M}$ if and only if $I \cup J \models \Sigma$.

Let $\mathcal{M}$ be a fixed schema mapping. In data exchange, the main problem is, given a source instance I, to find a *solution* for I w.r.t. $\mathcal{M}$, that is, a target instance J such that $(I, J) \in \mathcal{M}$ (or determine that no solution exists). We use the notation $\mathsf{Sol}(I, \mathcal{M}) = \{J \mid (I, J) \in \mathcal{M}\}$ to denote the set of all solutions for I w.r.t. $\mathcal{M}$. In data integration, the main problem is to compute the *certain answers* of queries [12]. Specifically, given a query q over the target schema and a source instance I, the *certain answers of q on I w.r.t. $\mathcal{M}$* is the set

$$\mathrm{cert}(q, I, \mathcal{M}) = \bigcap \{q(J) \mid J \in \mathsf{Sol}(I, \mathcal{M})\} .$$

On the face of it, the definition of certain answers may entail computing an intersection of infinitely many sets. One of the main findings in [6] is that there is a notion of a "good" solution in data exchange, called *universal solution*, that can also be used to compute the certain answers of conjunctive queries in a much more direct way.

Let J_1 and J_2 be two target instances. A function h is a *homomorphism* from J_1 to J_2 if the following hold: (i) for every constant c, we have that $h(c) = c$; and (ii) for every relation symbol R in $\mathbf{R}$ and every tuple $(a_1, \ldots, a_n) \in R^{J_1}$, we have that $(h(a_1), \ldots, h(a_n)) \in R^{J_2}$. We write $J_1 \to J_2$ to denote that there is a homomorphism from J_1 to J_2. We say that J_1 is *homomorphically equivalent* to J_2, written $J_1 \leftrightarrow J_2$, if $J_1 \to J_2$ and $J_2 \to J_1$.

Let I be a source instance. A *universal* solution for I w.r.t. $\mathcal{M}$ is a solution J such that for every solution $J' \in \mathsf{Sol}(I, \mathcal{M})$, we have that $J \to J'$. Intuitively, a universal solution for I is a "most general" solution for I. We write $\mathsf{UnivSol}(I, \mathcal{M})$ to denote the set of all universal solutions for I w.r.t. $\mathcal{M}$ (universal solutions need not always exist). As shown in [6], if q is a conjunctive query, I is a source instance, and J is a universal solution for I w.r.t. $\mathcal{M}$, then $\mathrm{cert}(q, I, \mathcal{M}) = q(J)_{\downarrow}$, where $q(J)_{\downarrow}$ is the set of all null-free tuples in $q(J)$.

Structural Properties of Schema Mappings. We now present a number of structural properties that a schema mapping may or may not possess. These properties were investigated in their own right in [19], where they were used to obtain characterizations of schema-mapping languages that will be of great interest to us in this paper. Let $\mathcal{M}$ be a schema mapping.
$\mathcal{M}$ *allows for* CQ*-rewriting* if for every target conjunctive query q, there exists a union q' of source conjunctive queries such that $\mathrm{cert}(I, \mathcal{M}, q) = q'(I)$, for every source instance I.
$\mathcal{M}$ *admits universal solutions* if for every source instance I, there is a universal solution for I w.r.t. $\mathcal{M}$. We write $\mathrm{univ}(I, \mathcal{M})$ to denote some such universal solution.
$\mathcal{M}$ is *closed under target homomorphisms* if $(I, J) \in \mathcal{M}$ and $J \to J'$ imply that $(I, J') \in \mathcal{M}$.
$\mathcal{M}$ is *closed under unions* if $(I_1, J_1) \in \mathcal{M}$ and $(I_2, J_2) \in \mathcal{M}$ imply that $(I_1 \cup I_2, J_1 \cup J_2) \in \mathcal{M}$.
$\mathcal{M}$ is *closed under target intersections* if $J_1 \in \mathsf{Sol}(I, \mathcal{M})$ and $J_2 \in \mathsf{Sol}(I, \mathcal{M})$ imply that $(J_1 \cap J_2) \in \mathsf{Sol}(I, \mathcal{M})$.
$\mathcal{M}$ is *n-modular* if whenever $(I, J) \notin \mathcal{M}$, there is a subinstance $I' \subseteq I$ with at most n elements in its active domain such that $(I', J) \notin \mathcal{M}$ ("small counterexample").

Schema Mapping Languages. A GLAV *(global-and-local-as-view)* constraint is a first-order formula of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x},\mathbf{y}))$, where $\varphi(\mathbf{x})$ is a conjunction of atoms over the source schema $\mathbf{S}$, each variable in $\mathbf{x}$ occurs in at least one atom in $\varphi(\mathbf{x})$, and $\psi(\mathbf{x},\mathbf{y})$ is a conjunction of atoms over the target schema $\mathbf{T}$ with variables in $\mathbf{x}$ and $\mathbf{y}$. We refer to $\varphi(\mathbf{x})$ as the *left-hand side*, or *premise*, and $\exists \mathbf{y}\psi(\mathbf{x},\mathbf{y})$ as the *right-hand side*, or *conclusion* of the constraint. Another name for GLAV constraints is *source-to-target tuple-generating dependencies* or, in short, *s-t tgds*.

A LAV (*local-as-view*) constraint is a GLAV constraint whose left-hand side is a single atom over the source, while a GAV (*global-as-view*) constraint is a GLAV constraint whose right-hand side is a single atom over the target (in particular, the right-hand side contains no existential quantifiers). For example, $\forall x,y(E(x,y) \rightarrow \exists z(F(x,z) \wedge F(z,y)))$ is a LAV constraint, and $\forall x,y,z(E(x,z) \wedge E(z,y) \rightarrow F(x,y))$ is a GAV constraint.

A GLAV (*global-and-local-as=view*) mapping is a schema mapping $\mathcal{M} = (\mathbf{S},\mathbf{T},\Sigma)$ such that Σ is a finite set of GLAV constraints. The notions of a *LAV mapping* and of a *GAV mapping* are defined analogously.

Every GLAV mapping $\mathcal{M}$ admits universal solutions [6]; furthermore, given a source instance I, a *canonical universal solution* $\text{chase}(I,\mathcal{M})$ can be produced via the *oblivious chase procedure* as follows: whenever the antecedent of an s-t tgd in $\mathcal{M}$ becomes true, fresh null values are introduced and facts involving these nulls are added to $\text{chase}(I,\mathcal{M})$, so that the conclusion of the s-t tgd becomes true. Every GLAV mapping is also known to allow for CQ-rewriting and to be n-modular, for some $n \geq 1$. Moreover, every LAV mapping is closed under unions, while every GAV mapping is closed under target intersections.

Second-Order tgds, or *SO tgds*, were introduced in [8] and were shown to be exactly the constraints needed to express the composition of a finite number of GLAV mappings. Instead of giving the precise definition of an SO tgd, we illustrate this notion with an example from [8]. The formula $\exists f(\forall e(Emp(e) \rightarrow Mgr(e,f(e))) \wedge \forall e(Emp(e) \wedge (e = f(e)) \rightarrow SelfMgr(e)))$ expresses the property that every employee has a manager, and if an employee is the manager of himself/herself, then this employee is a self-manager. The above formula is an SO tgd that is not logically equivalent to any (finite or infinite) set of GLAV constraints [8].

Every SO tgd allows for CQ-rewriting and admits universal solutions; however, an SO tgd may not be closed under target homomorphisms and there may not exist any $n \geq 1$ such that the SO tgd is n-modular (see [8, 19]).

Pseudometric Spaces and Metric Spaces. A *pseudometric space* is a pair (X,d), where X is a set and d is a function from $X \times X$ to the set R^+ of non-negative real numbers with the following properties: (i) $d(x,x) = 0$, for every x in X; (ii) $d(x,y) = d(y,x)$, for every x and y in X; (iii) $d(x,y) \leq d(x,z) + d(y,z)$, for every x, y, z in X (triangle inequality). A *metric space* is a pseudometric space (X,d) such that if $d(x,y) = 0$, then $x = y$. It is easy to see that if (X,d) is a pseudometric space, then the relation $R_d = \{(x,y) \in X \times X \mid d(x,y) = 0\}$ is an equivalence relation on X. From this, it follows that every pseudometric space (X,d) gives rise to a metric space $(\widehat{X},\widehat{d})$, where $\widehat{X}$ is the set of equivalence classes of elements of X modulo the equivalence relation R_d and $\widehat{d}([x],[y]) = d(x,y)$.

A sequence of elements $x_1,x_2,\ldots$ of X *converges* to an element x of X, denoted by $\lim_{n\to\infty} x_n = x$, if for every $\epsilon > 0$, there is an integer n_0 such that $d(x_n,x) < \epsilon$, for every $n \geq n_0$. We say that x is the *limit* of this sequence (the limit is unique if (X,d) is a metric space). A sequence $x_1,x_2,\ldots$ of elements of X is *Cauchy* if for every $\epsilon > 0$, there is an integer n_0 such that $d(x_n,x_{n'}) < \epsilon$, for every $n,n' \geq n_0$.

Using the triangle inequality, it is easy to see that if a sequence of elements in a (pseudo)metric space has a limit, then the sequence is Cauchy. The converse, however, does

not hold for arbitrary (pseudo)metric spaces. A (pseudo)metric space (X, d) is *complete* if every Cauchy sequence of elements of X has a limit in X; otherwise, it is *incomplete*.

It is well known that every incomplete (pseudo)metric space (X, d) can be embedded into a complete (pseudo)metric space (X^*, d^*), called the *completion* of (X, d), in such a way that X is a *dense* subset of X^*, i.e., every member of X^* is the limit of a sequence of members of X. The members of X^* are equivalence classes of Cauchy sequences of X, where two Cauchy sequences $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ of elements of X are *equivalent* if $\lim_{n\to\infty} d(x_n, y_n) = 0$, while the distance function d^* is defined as $d^*([x_1, x_2, \ldots], [y_1, y_2, \ldots]) = \lim_{n\to\infty} d(x_n, y_n)$. The proof of correctness of this construction can be found in [18] or any other book on metric spaces.

As a concrete example, the metric space of the real numbers is the completion of the metric space of the rational numbers (both with the standard distance).

3 Metric Space of Target Instances

To study the limits of sequences of schema mappings, we first introduce a pseudometric space of sets of target instances. By considering schema mappings as functions that map each source instance to the set of its solutions, we can view sequences of schema mappings as *sequences of functions*. The (pointwise or uniform) limit of a sequence of schema mappings is then simply defined in the standard way as the limit of a sequence of functions taking values in a pseudometric space. Moreover, by passing to the associated metric space of equivalence classes of sets of target instances, we ensure the uniqueness of the limit. If $\mathbf{T}$ is a schema, we write $\mathrm{Inst}(\mathbf{T})$ for the set of all finite instances of $\mathbf{T}$. We also write $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ for the power set of $\mathrm{Inst}(\mathbf{T})$. The notion of distance on $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ that we are about to introduce is heavily based on the notion of the certain answers to conjunctive queries and on the idea that two members $\mathcal{J}$ and $\mathcal{J}'$ of $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ are "close" to each other if only "big" conjunctive queries can yield different certain answers on $\mathcal{J}$ and $\mathcal{J}'$.

▶ **Definition 1.** Let q be a query over a schema $\mathbf{T}$ and let $\mathcal{J}$ be a member of $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$. The *certain answers* of q over $\mathcal{J}$ are defined as $\mathrm{cert}(q, \mathcal{J}) = \bigcap\{q(J) \mid J \in \mathcal{J}\}$.

We say that two sets of instances $\mathcal{J}$ and $\mathcal{J}'$ in $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ are CQ-*equivalent*, denoted $\mathcal{J} \equiv_{\mathsf{CQ}} \mathcal{J}'$, if $\mathrm{cert}(q, \mathcal{J}) = \mathrm{cert}(q, \mathcal{J}')$ for all conjunctive queries q.

We say that $\mathcal{J}$ and $\mathcal{J}'$ are CQ_n-*equivalent*, denoted $\mathcal{J} \equiv_{\mathsf{CQ}_n} \mathcal{J}'$, if $\mathrm{cert}(q, \mathcal{J}) = \mathrm{cert}(q, \mathcal{J}')$ for all conjunctive queries q with at most n variables (i.e., for all q in CQ_n.) ◁

▶ **Definition 2.** Let $\mathcal{J}$ and $\mathcal{J}'$ be two sets of instances in $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$. The *similarity* $\mathrm{sim}(\mathcal{J}, \mathcal{J}')$ and the *distance* $\mathrm{dist}(\mathcal{J}, \mathcal{J}')$ between $\mathcal{J}$ and $\mathcal{J}'$ are defined as follows:

- $\mathrm{sim}(\mathcal{J}, \mathcal{J}') = \max\{k \mid \mathcal{J} \equiv_{\mathsf{CQ}_k} \mathcal{J}'\}$;
- $\mathrm{dist}(\mathcal{J}, \mathcal{J}') = 2^{-\mathrm{sim}(\mathcal{J}, \mathcal{J}')}$. ◁

It is easy to verify that the pair $(\mathcal{P}(\mathrm{Inst}(\mathbf{T})), \mathrm{dist})$ is a pseudometric space; in fact, dist is an *ultrametric* distance function, that is, $\mathrm{dist}(\mathcal{J}, \mathcal{J}') \le \max\{\mathrm{dist}(\mathcal{J}, \mathcal{J}''), \mathrm{dist}(\mathcal{J}'', \mathcal{J}')\}$ holds for all $\mathcal{J}$, $\mathcal{J}'$, $\mathcal{J}''$ in $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$. Moreover, $\mathrm{dist}(\mathcal{J}, \mathcal{J}') = 0$ if and only if $\mathcal{J}$ and $\mathcal{J}'$ are CQ-equivalent. It is important to note that the pseudo-metric space $(\mathcal{P}(\mathrm{Inst}(\mathbf{T})), \mathrm{dist})$ is incomplete, i.e., there exist Cauchy sequences of elements of $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ that do not have a limit in $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$. We first give an example of a sequence that has a limit in $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$.

▶ **Example 3.** Let $\mathbf{T}$ be a schema consisting of a single binary relation and let C_m be the undirected cycle of length m, for $m \ge 1$. Consider the sequence $(\{C_{2n+1}\})_{n\ge1}$ of singletons each containing a cycle of odd size. It is not hard to verify that $\lim_{n\to\infty} (\{C_{2n+1}\})_{n\ge1} = \{C_2\}$. ◁

In contrast, there are Cauchy sequences of element of $\mathcal{P}(\text{Inst}(\mathbf{T}))$ that have no limit.

▶ **Proposition 4.** *Let $\mathbf{T}$ be a schema consisting of a single binary relation and let K_n be the clique of size n, for $n \geq 1$. The sequence $(\{K_n\})_{n\geq 1}$ of singletons each containing a clique of different size is Cauchy, but has no limit in $\mathcal{P}(\text{Inst}(\mathbf{T}))$.*

Proof. The sequence $(\{K_n\})_{n\geq 1}$ is Cauchy because if $m \geq n$, then K_m and K_n satisfy the same first-order sentences with n variables. To show that this sequence has no limit in $\mathcal{P}(\text{Inst}(\mathbf{T}))$, assume that there is a set $\mathcal{J}$ of finite instances over $\mathbf{T}$ such that $\lim_{n\to\infty}\{K_n\} = \mathcal{J}$. We distinguish two cases. If $\mathcal{J} = \varnothing$, then $\text{cert}(q, \mathcal{J}) = \textit{true}$, for every conjunctive query q. In contrast, $\text{cert}(\exists x E(x,x), \{K_n\}) = \textit{false}$, for every $n \geq 2$. If $\mathcal{J} \neq \varnothing$, consider a member J of $\mathcal{J}$. Let m be the biggest integer such that J contains a clique of size m, and let $\exists K_{m+1}$ be the conjunctive query asserting that there is a clique of size $m+1$. We now have that $\text{cert}(\exists K_{m+1}, \mathcal{J}) = \textit{false}$, while $\text{cert}(\exists K_{m+1}, \{K_n\}) = \textit{true}$, for every $n \geq m+1$. ◀

Since $(\{K_n\})_{n\geq 1}$ is a Cauchy sequence, it has a limit in the completion of $(\mathcal{P}(\text{Inst}(\mathbf{T})), \text{dist})$. A concrete representation of this limit is the singleton $\{K_\infty\}$, where K_∞ is the infinite clique. In Section 6, we will examine the completion of $(\mathcal{P}(\text{Inst}(\mathbf{T})), \text{dist})$ more closely.

The following definitions are perfectly meaningful for every pseudometric space (X, d) and for every sequence of functions taking values in X. For concreteness, we give the definitions for sequences of functions taking values in $\mathcal{P}(\text{Inst}(\mathbf{T}))$.

▶ **Definition 5.** Let A be a set, let $(f_n)_{n\geq 1}$ be a sequence of functions from A to $\mathcal{P}(\text{Inst}(\mathbf{T}))$, and let f be a function from A to $\mathcal{P}(\text{Inst}(\mathbf{T}))$.

- We say that $(f_n)_{n\geq 1}$ *converges pointwise* to f, denoted as $\lim^{p}_{n\to\infty} f_n = f$, if for every element $x \in A$, we have that $\lim_{n\to\infty} f_n(x) = f(x)$.
- We say that $(f_n)_{n\geq 1}$ *converges uniformly* to f, denoted as $\lim^{u}_{n\to\infty} f_n = f$, if for every $\epsilon > 0$, there exists an integer $n_0 \geq 1$ such that for every integer $n \geq n_0$ and for every element $x \in A$, we have $\text{dist}(f_n(x), f(x)) < \epsilon$.
- We say that $(f_n)_{n\geq 1}$ is *pointwise Cauchy*, if for every element $x \in A$, the sequence $(f_n(x))_{n\geq 1}$ is Cauchy.
- We say that $(f_n)_{n\geq 1}$ is *uniformly Cauchy*, if for every $\epsilon > 0$, there exists an integer $n_0 \geq 1$ such that for all integers $n, n' \geq n_0$ and for every element $x \in A$, we have $\text{dist}(f_n(x), f_{n'}(x)) < \epsilon$. ◁

Clearly, if $(f_n)_{n\geq 1}$ converges pointwise (resp., uniformly), then $(f_n)_{n\geq 1}$ is pointwise (resp., uniformly) Cauchy. The converse is not in general true for arbitrary (pseudo)metric spaces; in particular, it is not true for the pseudometric space $(\mathcal{P}(\text{Inst}(\mathbf{T})), \text{dist})$.

We now bring schema mappings into the picture. Every schema mapping $\mathcal{M}$ over a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$ can be identified with a function $f\colon \text{Inst}(\mathbf{S}) \longrightarrow \mathcal{P}(\text{Inst}(\mathbf{T}))$, where $f(I) = \mathsf{Sol}(I, \mathcal{M})$ (recall that $\mathsf{Sol}(I, \mathcal{M})$ is the set of all solutions of I w.r.t. $\mathcal{M}$, i.e., the set of all finite $\mathbf{T}$ instances J such that $(I, J) \in \mathcal{M}$). Thus, a sequence $(\mathcal{M}_n)_{n\geq 1}$ of schema mappings over a source schema $\mathbf{S}$ and target schema $\mathbf{T}$ can be viewed as a sequence of functions from $\text{Inst}(\mathbf{S})$ to $\mathcal{P}(\text{Inst}(\mathbf{T}))$. Therefore, we can talk about a sequence of schema mappings being pointwise Cauchy and uniformly Cauchy if the sequence of the associated functions has these properties. Similarly, we say that a sequence of schema mappings has a pointwise limit (resp., a uniform limit) if the sequence of the associated functions converges pointwise (resp., converges uniformly) to a schema mapping.

The preceding notion of convergence of a sequence of schema mappings allows us to draw a connection to earlier work on schema mapping optimization [5, 7]. Here, we are considering CQ-equivalence and CQ_n-equivalence of *sets of instances*. In previous works, these notions of equivalence have been mainly applied to schema mappings (see, e.g., [5, 7, 14]). Specifically, two schema mappings $\mathcal{M}, \mathcal{M}'$ are CQ-equivalent (resp., CQ_n-equivalent) if for every target conjunctive query q (resp., every target conjunctive query q in CQ_n) and every source instance I, we have that $\mathrm{cert}(q, I, \mathcal{M}) = \mathrm{cert}(q, I, \mathcal{M}')$. In this case, we write $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ (resp., $\mathcal{M} \equiv_{\mathsf{CQ}_n} \mathcal{M}'$). The notion of CQ_n-equivalence has been studied in the context of schema mapping optimization [5, 7]. Below we discuss its relationship to the convergence of schema mappings.

▶ **Proposition 6.** *Consider a sequence $(\mathcal{M}_n)_{n\geq 1}$ of schema mappings and a schema mapping $\mathcal{M}$. Then $\lim\limits_{n\to\infty}^{u} \mathcal{M}_n = \mathcal{M}$ if and only if for every integer $k \geq 1$, there is an integer $n_0 \geq 1$ such that for all integers $n \geq n_0$, we have that $\mathcal{M}_n \equiv_{\mathsf{CQ}_k} \mathcal{M}$.* ◁

Intuitively, the preceding proposition states that it takes bigger and bigger conjunctive queries to distinguish the members of a sequence $(\mathcal{M}_n)_{n\geq 1}$ from its uniform limit.

Although never explicitly introduced, the notion of uniform convergence was implicit in [5], where it was shown that for every SO tgd σ and for every $n \geq 1$, there is a GLAV mapping $\mathcal{M}_n$ such that $\sigma \equiv_{\mathsf{CQ}_n} \mathcal{M}_n$. From this, it is easy to see that $\lim\limits_{n\to\infty}^{u} \mathcal{M}_n = \sigma$. Thus, we have the following result.

▶ **Theorem 7** (implicit in [5])**.** *Every SO tgd is a uniform limit of a sequence of GLAV mappings.*

There are SO tgds that are not CQ-equivalent to any GLAV mapping [7]. Thus, the point of Theorem 7 is that SO tgds can be "approximated" up to any level of CQ_k-equivalence by GLAV mappings, which are syntactically simpler and generally more well-behaved.

As stated earlier, $(\mathcal{P}(\mathrm{Inst}(\mathbf{T})), \mathrm{dist})$ is a pseudometric space since it cannot distinguish CQ-equivalent sets of instances. Consequently, the limit of a sequence of sets of instances and the (uniform or pointwise) limit of a sequence of mappings need not be unique. However, the limit is unique up to CQ-equivalence and, as described in Section 2, there is an associated metric space $(\mathcal{P}(\widehat{\mathrm{Inst}(\mathbf{T})}), \widehat{\mathrm{dist}})$ obtained by considering the equivalence classes of $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ modulo the equivalence relation R_{dist}, where $(\mathcal{J}, \mathcal{J}') \in R_{\mathrm{dist}}$ if and only if $\mathrm{dist}(\mathcal{J}, \mathcal{J}') = 0$ (i.e., if and only if $\mathcal{J} \equiv_{\mathsf{CQ}} \mathcal{J}'$).

In subsequent sections, we will work with the metric space $(\mathcal{P}(\widehat{\mathrm{Inst}(\mathbf{T})}), \widehat{\mathrm{dist}})$. Moreover, we will be interested in schema mappings modulo CQ-equivalence, which means that from now on we will view schema mappings as functions from source instances to equivalence classes of sets of target instances modulo CQ-equivalence. However, for notational simplicity, we will work each time with representatives of the equivalence classes. By a slight abuse of notation, we will write $(\mathcal{P}(\mathrm{Inst}(\mathbf{T})), \mathrm{dist})$, instead of $(\mathcal{P}(\widehat{\mathrm{Inst}(\mathbf{T})}), \widehat{\mathrm{dist}})$. Likewise, we will not explicitly distinguish between a schema mapping $\mathcal{M}$ and the equivalence class of schema mappings that are CQ-equivalent to $\mathcal{M}$.

4 Limits of Sequences of GAV Mappings

Our goal in this section is to analyze sequences of GAV mappings. To this effect, we first investigate the existence of limits of such sequences and then examine the definability of limits. As discussed in Section 3, if a sequence $(\mathcal{M}_n)_{n\geq 1}$ of schema mappings has a pointwise

(resp., uniform) limit, then the sequence is pointwise (resp., uniformly) Cauchy. The next result asserts that the converse holds for sequences of GAV mappings.

▶ **Theorem 8.** *Let $(\mathcal{M}_n)_{n\geq 1}$ be a sequence of GAV mappings.*
- *If $(\mathcal{M}_n)_{n\geq 1}$ is pointwise Cauchy, then it has a pointwise limit.*
- *If $(\mathcal{M}_n)_{n\geq 1}$ is uniformly Cauchy, then it is eventually constant and thus has a GAV schema mapping as a uniform limit.*

Proof Sketch. For showing the first claim, assume that $(\mathcal{M}_n)_{n\geq 1}$ is a pointwise Cauchy sequence of schema mappings and let I be a source instance. For each $n \geq 1$, consider the universal solution $\text{chase}(I, \mathcal{M}_n)$ for I w.r.t. $\mathcal{M}_n$ obtained by using the oblivious chase procedure. Since each $\mathcal{M}_n$ is a GAV schema mapping, we have that $\text{chase}(I, \mathcal{M}_n)$ contains no nulls. It can be shown that there exists some m_I s.t. for all $n \geq m_I$, we have that $\text{chase}(I, \mathcal{M}_n) = \text{chase}(I, \mathcal{M}_{m_I})$. In other words, the sequence $(\text{chase}(I, \mathcal{M}_n))_{n\geq 1}$ is eventually constant (does not oscillate). Then the schema mapping $\mathcal{M} = \{(I, \text{chase}(I, \mathcal{M}_{m_I})) \mid I \text{ is a source instance}\}$ is a pointwise limit of the sequence $(\mathcal{M}_n)_{n\geq 1}$.

For showing the second claim, assume that $(\mathcal{M}_n)_{n\geq 1}$ is a uniformly Cauchy sequence of GAV mappings. We claim that $(\mathcal{M}_n)_{n\geq 1}$ is eventually constant, i.e., there is some m such that for all $n \geq m$, $\mathcal{M}_n \equiv_{\mathsf{CQ}} \mathcal{M}_m$ holds. Towards a contradiction, assume that for every m there exists an $i > m$ such that $\mathcal{M}_i \not\equiv_{\mathsf{CQ}} \mathcal{M}_m$. That is, for some source instance I, it is the case that $\text{chase}(I, \mathcal{M}_m) \neq \text{chase}(I, \mathcal{M}_i)$. Since neither $\text{chase}(I, \mathcal{M}_m)$ nor $\text{chase}(I, \mathcal{M}_i)$ contain nulls, they can be distinguished using atomic queries from CQ_k, where k is the maximum relation arity of the target schema. Since this is the case for an arbitrarily large m, it follows that $(\mathcal{M}_n)_{n\geq 1}$ is not a uniformly Cauchy sequence, a contradiction. ◀

Next, we point out that even simple sequences of GAV schema mappings may have no uniform limit.

▶ **Proposition 9.** *There exists a sequence of GAV mappings that has a pointwise limit but no uniform limit.*

Proof. For every $n \geq 2$, let $\exists K_n$ be the boolean conjunctive query asserting that there is a clique of size n, i.e., $\exists K_n$ is the expression $\exists x_1, \ldots x_n \bigwedge_{i\neq j}(E(x_i, x_j) \wedge E(x_j, x_i))$.

Let $(\mathcal{M}_n)_{n\geq 1}$ be the sequence of GAV mappings, where $\mathcal{M}_n$ is specified by the constraint $\forall x(P(x) \wedge \exists K_{n+1} \rightarrow P'(x))$. Intuitively, $\mathcal{M}_n$ is a "copy" schema mapping, but the copying action is triggered only if E contains a clique of size $n+1$. One can show that the GAV schema mapping $\mathcal{M} = \{\forall x \forall y(P(x) \wedge E(y, y) \rightarrow P'(x))\}$ is a pointwise limit of $(\mathcal{M}_n)_{n\geq 1}$, but that this pointwise limit is not a uniform limit of $(\mathcal{M}_n)_{n\geq 1}$ and thus no uniform limit of $(\mathcal{M}_n)_{n\geq 1}$ exists.

To see that $\mathcal{M}$ is a pointwise limit of $(\mathcal{M}_n)_{n\geq 1}$, note that for source instances with a self-loop $E(a, a)$ for some a, $\mathcal{M}$ is indistinguishable from every element $\mathcal{M}_i \in (\mathcal{M}_n)_{n\geq 1}$. For source instances without such a self-loop, $\mathcal{M}$ coincides with all members of $(\mathcal{M}_n)_{n\geq 1}$ with an index exceeding the size of the maximal clique in I.

Now towards a contradiction assume that $\mathcal{M}$ is also a uniform limit. Then, there must be an n_0 such that for all $n \geq n_0$, the equivalence $\mathcal{M}_n \equiv_{\mathsf{CQ}_1} \mathcal{M}$ holds. However, taking $n = n_0$ and a source instance $I = K_n \cup \{P(c)\}$, one can observe that a target CQ_1 query $q = \exists x\, P'(x)$ witnesses $\mathcal{M}_n \not\equiv_{\mathsf{CQ}_1} \mathcal{M}$, since I contains no self-loop and thus $\mathsf{UnivSol}(I, \mathcal{M}) = \{\varnothing\}$. ◀

Proposition 9 and Theorem 8 imply that the sequence of GAV mappings in the proof of Proposition 9 is an example of a pointwise Cauchy sequence that is not uniformly Cauchy. More importantly, Theorem 8 gives rise to the following natural question concerning the

definability of limits: if a sequence of GAV mappings has a pointwise limit, does it have a GAV mapping as such a limit? We answer this question in the negative: even the much richer language of SO tgds cannot express pointwise limits of sequences of GAV mappings.

▶ **Proposition 10.** *There is a pointwise Cauchy sequence of GAV schema mappings such that no SO tgd is a pointwise limit of that sequence.*

Proof Idea. For every $n \geq 1$, let $P_n(x, y)$ be the conjunctive query expressing the property "there is an E-path of length n from x to y", and let $\mathcal{M}_n$ be the GAV mapping specified by the set $\{\forall x, y(P_i(x, y) \rightarrow F(x, y)) \mid 1 \leq i \leq n\}$. The schema mapping

$$\mathcal{M}^\star = \{(I, J) \mid F^J \text{ contains the transitive closure } TC(I) \text{ of } E^I\}$$

is a pointwise limit of the sequence $(\mathcal{M}_n)_{n\geq 1}$. However, note that $\mathcal{M}^\star$ is not CQ-equivalent to any schema mapping $\mathcal{M}'$ that allows for CQ-rewriting: if it were, then there would exist a union q of conjunctive queries over the source such that, for every source instance I, $\text{cert}(F(x, y), I, \mathcal{M}^*) = TC(I) = \text{cert}(F(x, y), I, \mathcal{M}') = q(I)$. Consequently, the transitive closure of I would be first-order definable over the source, which is not the case. Since every SO tgd allows for CQ-rewriting, no SO tgd is a pointwise limit of the sequence $(\mathcal{M}_n)_{n\geq 1}$. ◀

We have just seen that there are sequences of GAV mappings that have a pointwise limit, but no such limit is definable by a GAV mapping. This raises the question of finding necessary and sufficient conditions guaranteeing that a sequence of GAV mappings has a GAV mapping as a pointwise limit. The next result provides an answer to this question.

▶ **Theorem 11.** *Let $(\mathcal{M}_n)_{n\geq 1}$ be a pointwise Cauchy sequence of GAV mappings. The following statements are equivalent:*
1. *$(\mathcal{M}_n)_{n\geq 1}$ has a GAV mapping as a pointwise limit.*
2. *$(\mathcal{M}_n)_{n\geq 1}$ has a pointwise limit that allows for CQ-rewriting.*

Proof Idea. Let $(\mathcal{M}_n)_{n\geq 1}$ be a pointwise Cauchy sequence of schema mappings. As seen in the proof sketch of Theorem 8, for every source instance I, there is a positive integer m_I, such that for all $n \geq m_I$ the equality $\text{chase}(I, \mathcal{M}_{m_I}) = \text{chase}(I, \mathcal{M}_n)$ holds for the respective elements $\mathcal{M}_{m_I}$ and $\mathcal{M}_n$ of $(\mathcal{M}_n)_{n\geq 1}$. Moreover, the schema mapping $\mathcal{M} = \{(I, \text{chase}(I, \mathcal{M}_{m_I}) \mid I \text{ is a source instance}\}$ is a pointwise limit of $(\mathcal{M}_n)_{n\geq 1}$, and so is the CQ-equivalent mapping $\mathcal{M}^\star = \{(I, J) \mid \text{chase}(I, \mathcal{M}_{m_I}) \subseteq J\}$. The result we seek is an immediate consequence of the fact that the following four statements are equivalent:
(a) $(\mathcal{M}_n)_{n\geq 1}$ has a GAV mapping as a pointwise limit.
(b) $(\mathcal{M}_n)_{n\geq 1}$ has a pointwise limit that allows for CQ-rewriting.
(c) $\mathcal{M}^\star$ allows for CQ-rewriting.
(d) $\mathcal{M}^\star$ is logically equivalent to a GAV mapping.
The proof uses Theorem 3.2 in [19], which asserts that a schema mapping is logically equivalent to a GAV schema mapping if and only if it allows for CQ-rewriting, admits universal solutions, and is closed under both target homomorphisms and target intersections. ◀

▶ **Corollary 12.** *Let $(\mathcal{M}_n)_{n\geq 1}$ be a pointwise Cauchy sequence of GAV mappings. The following statements are equivalent:*
1. *$(\mathcal{M}_n)_{n\geq 1}$ has a GAV mapping as a pointwise limit.*
2. *$(\mathcal{M}_n)_{n\geq 1}$ has an SO tgd as a pointwise limit.*

Proposition 10 and Theorem 11 yield a fairly complete picture of the definability of pointwise limits of GAV mappings. Specifically, there are two mutually exclusive possibilities:

1. No pointwise limit allows for CQ-rewriting and no GAV mapping is a pointwise limit.
2. Every pointwise limit admits CQ-rewriting and there is a GAV mapping that is a pointwise limit. Moreover, this happens precisely when the schema mapping $\mathcal{M}^\star$ in the proof of Theorem 11 allows for CQ-rewriting or, equivalently, when $\mathcal{M}^\star$ is logically equivalent to a GAV mapping.

5 Limits of Sequences of LAV Mappings

In this section, we investigate the existence and definability of limits of sequences of LAV mappings. In fact, we will consider a much broader class of GLAV mappings than LAV, which we call *premise-bounded* GLAV mappings. LAV mappings are the special case of this class when the premise bound is equal to one.

▶ **Definition 13.** Let $(\mathcal{M}_n)_{n\geq 1}$ be a sequence of GLAV mappings. We say that $(\mathcal{M}_n)_{n\geq 1}$ is *premise-bounded* if there exists an integer k such that for every element $\mathcal{M}_n$ of $(\mathcal{M}_n)_{n\geq 1}$, the premise of every constraint in $\mathcal{M}_n$ has at most k atoms.

Unlike the case of GAV mappings, the notions of pointwise Cauchy and uniformly Cauchy sequences of premise-bounded GLAV mappings coincide. Moreover, the same holds true for the notions of pointwise limit and uniform limit of sequences of such schema mappings.

▶ **Theorem 14.** *Let $(\mathcal{M}_n)_{n\geq 1}$ be a sequence of premise-bounded GLAV mappings.*
1. *The sequence $(\mathcal{M}_n)_{n\geq 1}$ is pointwise Cauchy if and only if it is uniformly Cauchy.*
2. *The sequence $(\mathcal{M}_n)_{n\geq 1}$ has a pointwise limit if and only if it has a uniform limit.*

The following two propositions further demarcate the differences between GAV and premise-bounded mappings. In fact, these differences are already witnessed by sequences of LAV mappings. The first difference concerns the existence of limits of uniformly Cauchy sequences. In contrast to the GAV case, uniformly Cauchy sequences of LAV mappings may have no uniform limit; in fact, they may not even have a pointwise limit.

▶ **Proposition 15.** *There exists a uniformly Cauchy sequence of LAV mappings that has no pointwise limit; in particular, it has no uniform limit either.*

Proof Idea. For every $n \geq 1$, let $\mathcal{M}_n$ be the LAV mapping specified by the constraint $\forall x, y(E(x,y) \rightarrow \exists K_{n+1})$, where, as earlier, $\exists K_{n+1}$ is the boolean conjunctive query asserting that there is a clique of size $n+1$. Using an argument similar to the one in the proof of Proposition 4, it can be shown that the sequence $(\mathcal{M}_n)_{n\geq 1}$ has no pointwise limit. ◀

The next difference is the definability of uniform limits. In Section 4, we saw that if a sequence of GAV mappings has a uniform limit, then it is eventually constant, hence it has a GAV mapping as a uniform limit. This property need not hold for sequences of LAV mappings (hence, it need not hold for sequences of premise-bounded schema mappings).

▶ **Proposition 16.** *There exists a sequence $(\mathcal{M}_n)_{n\geq 1}$ of LAV mappings that has a uniform limit, but no uniform limit of $(\mathcal{M}_n)_{n\geq 1}$ admits universal solutions. In particular, no SO tgd is a uniform limit of the sequence $(\mathcal{M}_n)_{n\geq 1}$.*

Proof Idea. For every $n \geq 1$, let $\mathcal{M}_n$ be the LAV mapping specified by the constraint $\forall x(V(x) \rightarrow \exists P_n)$, where $\exists P_n$ is a boolean CQ asking for a path of length n in the target instance. We argue that the mapping $\mathcal{M} = \{(\varnothing, \varnothing)\} \cup \{(I, C_k) \mid I \text{ non-empty and } k > 1\}$ is the uniform limit of $(\mathcal{M}_n)_{n\geq 1}$, and that $\mathcal{M}$ does not admit universal solutions. ◀

By Theorem 7, every SO tgd is the uniform limit of a sequence of GLAV mappings. Proposition 16 implies that the converse is false, even for sequences of LAV mappings.

In the previous section, we showed that a sequence of GAV mappings has a GAV mapping as a pointwise limit if and only if it has a pointwise limit that allows for CQ-rewriting. Is there some structural property that characterizes when a sequence of premise-bounded GLAV mappings has a GLAV mapping as a pointwise limit (which, for premise-bounded mappings, is the same as a uniform limit)? We will show that the property of admitting universal solutions is the key to this question. Specifically, we have the following result.

▶ **Theorem 17.** *Let $(\mathcal{M}_n)_{n\geq 1}$ be a premise-bounded sequence of GLAV mappings. The following statements are equivalent.*

1. *$(\mathcal{M}_n)_{n\geq 1}$ has a GLAV mapping $\mathcal{M}$ as a uniform limit.*
2. *$(\mathcal{M}_n)_{n\geq 1}$ has a uniform limit that admits universal solutions.*

Moreover, if $(\mathcal{M}_n)_{n\geq 1}$ is a sequence of LAV mappings, then $(\mathcal{M}_n)_{n\geq 1}$ has a LAV mapping as a uniform limit if and only $(\mathcal{M}_n)_{n\geq 1}$ has a uniform limit that admits universal solutions.

Proof (Hint). The direction (1) ⇒ (2) is obvious. For the direction (2) ⇒ (1), we start with the case when $(\mathcal{M}_n)_{n\geq 1}$ is a sequence of LAV mappings. As stepping stones to the proof, the following lemmas can be used, which are of interest in their own right.

▶ **Lemma 18.** *If $\mathcal{M}$ is the uniform limit of a sequence $(\mathcal{M}_n)_{n\geq 1}$ of schema mappings each of which allows for CQ-rewriting, then also $\mathcal{M}$ allows for CQ-rewriting.*

▶ **Lemma 19.** *Let $\mathcal{M}$ be a uniform limit of a sequence $(\mathcal{M}_n)_{n\geq 1}$ of LAV mappings. If $\mathcal{M}$ admits universal solutions, then it is closed under unions.*

Assume that $\mathcal{M}$ is a uniform limit of a sequence $(\mathcal{M}_n)_{n\geq 1}$ of LAV mappings and that $\mathcal{M}$ admits universal solutions. Since the notion of limit is based on CQ-equivalence, we may assume w.l.o.g. that $\mathcal{M}$ is closed under target homomorphisms. Then $\mathcal{M}$ has the following properties: $\mathcal{M}$ admits universal solutions; $\mathcal{M}$ allows for CQ-rewriting (Lemma 18); $\mathcal{M}$ is closed under target homomorphisms; $\mathcal{M}$ is closed under unions (by Lemma 19). From Theorem 3.1 in [19], it follows that $\mathcal{M}$ is logically equivalent to a LAV mapping.

For the case when $(\mathcal{M}_n)_{n\geq 1}$ is a sequence of premise-bounded GLAV mappings (but not necessarily LAV mappings), we apply yet another structural characterization theorem from [19], namely Theorem 3.9, which asserts that if a schema mapping allows for CQ-rewriting, admits universal solutions, is closed under target homomorphisms, and is n-modular, for some fixed n, then it is logically equivalent to a GLAV mapping. Using machinery similar to the one used for the closure under unions in Lemma 19, it can be shown that the uniform limit $\mathcal{M}$ of the sequence $(\mathcal{M}_n)_{n\geq 1}$ is n-modular for some fixed $n \geq 1$. The other structural properties are handled as in the case of a sequence of LAV mappings. ◀

We conclude this section with a conjecture concerning uniform limits of arbitrary sequences of GLAV mappings.

▶ **Conjecture 20.** *The following statements are equivalent for a sequence $(\mathcal{M}_n)_{n\geq 1}$ of GLAV mappings.*

1. *$(\mathcal{M}_n)_{n\geq 1}$ has an SO tgd as a uniform limit.*
2. *$(\mathcal{M}_n)_{n\geq 1}$ has a uniform limit that admits universal solutions.*

It is not hard to show that the preceding conjecture is implied by a conjecture in [2] to the effect that the language of *plain*[2] SO-tgds can be characterized by the following three properties: allowing for CQ-rewriting, admitting universal solutions, and closure under target homomorphisms. It appears, however, that the technical tools needed to resolve the conjecture in [2] are not available at present.

6 Metric Space Completion and Generalized Schema Mappings

Let $\mathbf{T}$ be a schema containing a binary relation symbol. By Proposition 4, the metric space $(\mathcal{P}(\text{Inst}(\mathbf{T})), \text{dist})$ is not complete, i.e., there are Cauchy sequences of elements of $\mathcal{P}(\text{Inst}(\mathbf{T}))$ that have no limit in $\mathcal{P}(\text{Inst}(\mathbf{T}))$. Let $(\mathcal{P}(\text{Inst}(\mathbf{T}))^*, \text{dist}^*)$ be the completion of $(\mathcal{P}(\text{Inst}(\mathbf{T})), \text{dist})$. As described in Section 2, the elements of $\mathcal{P}(\text{Inst}(\mathbf{T}))^*$ are the equivalence classes of Cauchy sequences of elements of $\mathcal{P}(\text{Inst}(\mathbf{T}))$, where two Cauchy sequences $\mathcal{I}_1, \mathcal{I}_2, \ldots$ and $\mathcal{J}_1, \mathcal{J}_2, \ldots$ are equivalent if $\lim_{n\to\infty} \text{dist}(\mathcal{I}_n, \mathcal{J}_n) = 0$. Clearly, this is a rather abstract description of $\mathcal{P}(\text{Inst}(\mathbf{T}))^*$. The main result of this section reveals that the elements of $\mathcal{P}(\text{Inst}(\mathbf{T}))^*$ can be represented by suitably constructed infinite $\mathbf{T}$-instances. In turn, this result and basic results about complete metric spaces imply that the (pointwise or uniform) limits of a Cauchy sequence of schema mappings can be represented by a *generalized* schema mapping, that is, a schema mapping in which infinite solutions are allowed.

Let q be a conjunctive query with k free variables and let $\mathbf{a}$ be a k-tuple of constants. We write $q(\mathbf{a})$ to denote the instance K obtained by (i) substituting the free variables of q by the respective elements of $\mathbf{a}$; (ii) replacing the existential variables of q by fresh distinct labeled nulls; and (iii) treating the resulting body atoms of q as facts of the instance K.

We write $J_1 \uplus J_2$ to denote the *disjoint union* of two instances J_1 and J_2, that is, the instance obtained as a union of J_1 and J_2 with all labeled nulls renamed apart. If X is a set of instances, we write $\biguplus X$ to denote the disjoint union of all members of X. Note that we do not necessarily assume X to be finite; thus, $\biguplus X$ may be an infinite instance.

We are now ready to state the main result of this section and sketch its proof.

▶ **Theorem 21.** *Let $(\mathcal{J}_n)_{n\geq 1}$ be a Cauchy sequence of elements of $\mathcal{P}(\text{Inst}(\mathbf{T}))$. Then the limit of the sequence $(\mathcal{J}_n)_{n\geq 1}$ is the singleton $\mathbf{T}$-instance set $\mathcal{J}^*$, where*

$$\mathcal{J}^* = \Big\{ \biguplus \{ q(\mathbf{a}) \mid q \in \mathsf{CQ} \text{ and there is an integer } p \text{ such that } \mathbf{a} \in \text{cert}(q, \mathcal{J}_i), \text{ for every } i \geq p \} \Big\}.$$

Proof (Sketch). We have to show that $(\mathcal{J}_n)_{n\geq 1} \longrightarrow \mathcal{J}^*$, which means that for every integer $m \geq 1$ there exists an integer $n_0 \geq 1$ such that $\mathcal{J}_n \equiv_{\mathsf{CQ}_m} \mathcal{J}^*$, for all $n \geq n_0$.

By definition, $\mathcal{J}^*$ is a singleton; we write J to denote the single element of $\mathcal{J}^*$. The first crucial observation is that the set D of constants occurring in J is finite. To show this, we consider *single-atom* conjunctive queries, that is, queries of the form $\exists \mathbf{y} R(\mathbf{x}, \mathbf{y})$, where R is a relation symbol in the target schema $\mathbf{T}$. Clearly, every single-atom query has at most k variables, where k is the maximum arity of the relation symbols in $\mathbf{T}$.

Since the sequence $(\mathcal{J}_n)_{n\geq 1}$ is Cauchy, there exists an integer p_k such that $\mathcal{J}_i \equiv_{\mathsf{CQ}_k} \mathcal{J}_{p_k}$, for all $i \geq p_k$. This implies that the certain answers to single-atom conjunctive queries become fixed in $(\mathcal{J}_n)_{n\geq 1}$ starting from some integer p_k that depends only on the schema $\mathbf{T}$. By definition, the certain answers hold in every instance in $\mathcal{J}_{p_k}$; moreover, every instance in $\mathcal{J}_{p_k}$ is finite. Hence, the set D' of the certain answers to single-atom conjunctive queries

[2] A *plain* SO tgd is an SO tgd that contains no nested terms and no equalities. Every SO tgd is known to be CQ-equivalent to a plain one [2].

that eventually hold in $(\mathcal{J}_n)_{n\geq 1}$ is finite. To complete the proof of the finiteness, we show that the set D of constants occurring in the instance J is contained in D'. To see this, recall that J is composed of the bodies of conjunctive queries $q(\mathbf{a})$ such that $\mathbf{a} \in \mathrm{cert}(q, \mathcal{J}_n)$, for all sufficiently large n. Fix such a conjunctive query q with r atoms and consider its decomposition to single-atom queries $q_i(\mathbf{a}_i), \ldots, q_r(\mathbf{a}_r)$, where q_i has the i-th atom of q as its body and $\mathbf{a}_i$ contains the constants of $\mathbf{a}$ occurring in this atom. Observe that $\mathbf{a} \in \mathrm{cert}(q, \mathcal{J})$ implies $\mathbf{a}_i \in \mathrm{cert}(q_i, \mathcal{J})$, for every set $\mathcal{J}$ of instances. Consequently, the inclusion $D \subseteq D'$ holds, and thus D must be finite.

Now, given m, we need to provide n_0 such that for all $n \geq n_0$, we have that $\mathcal{J}_n \equiv_{\mathsf{CQ}_m} \mathcal{J}^*$ holds. In other words, n_0 has to big enough to ensure the equality $\mathrm{cert}(q, \mathcal{J}_n) = \mathrm{cert}(q, \mathcal{J}^*)$ for every conjunctive query $q \in \mathsf{CQ}_m$. In order to guarantee the inclusion $\mathrm{cert}(q, \mathcal{J}_n) \subseteq \mathrm{cert}(q, \mathcal{J}^*)$, it suffices to choose n_0 greater than the index n_1, starting from which all certain answers to CQ_m queries become fixed in $(\mathcal{J}_n)_{n\geq 1}$. Such an index n_1 exists since the sequence $(\mathcal{J}_n)_{n\geq 1}$ is Cauchy. To ensure $\mathrm{cert}(q, \mathcal{J}^*) \subseteq \mathrm{cert}(q, \mathcal{J}_n)$, we analyze the values of q in the limit instance J (recall that $\mathcal{J}^*$ is a singleton $\{J\}$). By the definition of J^*, atoms witnessing that $J \models q(\mathbf{b})$ stem from the bodies of conjunctive queries $q_1(\mathbf{a}_1), \ldots, q_\ell(\mathbf{a}_\ell)$. All these conjunctive queries hold in $(\mathcal{J}_n)_{n\geq 1}$ starting from some index. Inspecting finitely many conjunctive queries in CQ_m and all possible certain answers to them, one can choose n_0 large enough to ensure that $\mathrm{cert}(q, \mathcal{J}^*) \subseteq \mathrm{cert}(q, \mathcal{J}_n)$ as well. ◀

In their recent monograph [15], Nešetřil and Ossona de Mendez considered a notion of distance between instances, as well as sequences of instances and their limits. However, they considered a different setting and followed a different approach: first, they did not distinguish two classes of domain elements (constants and nulls) and, second, they heavily relied on a quasi-order on instances based on homomorphisms. The limit of a Cauchy sequence of instances is obtained in [15] via the concept of *ideal completion*. If $(\mathcal{J}_n)_{n\geq 1}$ is a Cauchy sequence of elements of $\mathcal{P}(\mathrm{Inst}(\mathbf{T}))$ such that all target instances appearing in this sequence contain only nulls (and no constants), then our description of the limit $\mathcal{J}*$ can be shown to be equivalent in the one in [15]; moreover, in this case, only boolean conjunctive queries contribute to the disjoint unions defining the limit.

We now recall two basic results about complete metric spaces.

▶ **Proposition 22.** *Let (Y, d) be a complete metric space and let $(f_n)_{n\geq 1}$ be a sequence of function from a set X to Y.*

- *If $(f_n)_{n\geq 1}$ is a pointwise Cauchy sequence, then $(f_n)_{n\geq 1}$ has a pointwise limit $f : X \to Y$, where $f(x) = \lim_{n\to\infty} f_n(x)$, for every $x \in X$.*
- *If $(f_n)_{n\geq 1}$ is a uniformly Cauchy sequence, then $(f_n)_{n\geq 1}$ has a uniform limit. Moreover, the pointwise limit $f : X \to Y$ of $(f_n)_{n\geq 1}$ is also the uniform limit of $(f_n)_{n\geq 1}$.*

The proof of the first part of Proposition 22 is immediate from the definitions; the proof of the second part can be found in any standard book on metric spaces (see, e.g., Proposition 3.6.6 in [18]). Note that the second part of Proposition 22 is known as the *Cauchy criterion*.

We are now ready to obtain concrete representations of the (pointwise or uniform) limits of Cauchy sequences of schema mappings.

▶ **Definition 23.** Let $\mathbf{S}, \mathbf{T}$ be two schemas. A *generalized schema mapping* is a set $\mathcal{M}$ of pairs (I, J) such that I is a finite $\mathbf{S}$-instance and J is a possibly infinite $\mathbf{T}$-instance.

▶ **Corollary 24.** *Let $(\mathcal{M}_n)_{n\geq 1}$ be a sequence of schema mappings. Consider the generalized schema mapping $\mathcal{M} = \big\{(I, J) \mid J = \biguplus\{q(\mathbf{a}) \mid q \in \mathsf{CQ} \text{ and } \exists p\ \forall i \geq p\ \mathbf{a} \in \mathrm{cert}(q, I, \mathcal{M}_i)\}\big\}$*

- *If* $(\mathcal{M}_n)_{n\geq 1}$ *is a pointwise Cauchy sequence, then the schema mapping* $\mathcal{M}$ *is the pointwise limit of* $(\mathcal{M}_n)_{n\geq 1}$.
- *If* $(\mathcal{M}_n)_{n\geq 1}$ *is a uniformly Cauchy sequence, then the schema mapping* $\mathcal{M}$ *is the uniform limit of* $(\mathcal{M}_n)_{n\geq 1}$.

Proof. The first part follows from Theorem 21 and the definition of pointwise convergence. The second part follows from the first part and Proposition 22. ◀

Finally, we consider (pointwise or uniformly) Cauchy sequences of schema mappings admitting universal solutions and obtain a different representation of their limits.

▶ **Corollary 25.** *Let* $(\mathcal{M}_n)_{n\geq 1}$ *be a pointwise Cauchy sequence of schema mappings over a source schema* $\mathbf{S}$ *and a target schema* $\mathbf{T}$*, each admitting universal solutions.*

1. *For every* $I \in Inst(\mathbf{S})$*, the sequence* $(\mathsf{UnivSol}(I, \mathcal{M}_n))_{n\geq 1}$ *is Cauchy, and hence it has a limit* $\lim_{n\to\infty}(\mathsf{UnivSol}(I, \mathcal{M}_n))$ *in the complete metric space* $(\mathcal{P}(\mathrm{Inst}(\mathbf{T}))^*, \mathrm{dist}^*)$.
2. *The generalized schema mapping* $\mathcal{M}^* = \{(I, J) \mid I \in Inst(\mathbf{S}), J \in \lim_{n\to\infty}(\mathsf{UnivSol}(I, \mathcal{M}_n))\}$ *is a pointwise limit of* $(\mathcal{M}_n)_{n\geq 1}$*. Moreover, if* $(\mathcal{M}_n)_{n\geq 1}$ *is a uniformly Cauchy sequence, then* $\mathcal{M}^*$ *is its uniform limit.*

7 Concluding Remarks

In this paper, we have embarked on a systematic study of the limiting behavior of sequences of schema mappings using concepts and tools from metric spaces. For the important special cases of GAV and LAV mappings, our main results are summarized in Figures 1 and 2.

In words, we have shown that, for GAV mappings, a pointwise Cauchy sequence need not be uniformly Cauchy; moreover, the existence of a pointwise limit does not imply the existence of a uniform limit. This cannot happen for LAV mappings. On the other side, a uniformly Cauchy sequence of LAV mappings need not even have a pointwise limit, which cannot happen for GAV mappings. We have also shown that structural properties of schema mappings can be used to characterize when the limit of a pointwise Cauchy sequence of GAV (or of LAV) mappings is equivalent to a GAV (or to a LAV) mapping. Finally, we have shown that infinite target instances and generalized mappings (i.e., schema mappings where target instances may be infinite) can be used to represent limits of Cauchy sequences of sets of target instances and limits of Cauchy sequences of arbitrary schema mappings.

We believe that the work reported here has laid the foundation for several interesting lines of subsequent investigations. We have seen that our results about sequences of LAV mappings extend in a natural way to sequences of premise-bounded GLAV mappings; an analogous extension of our results about sequences of GAV mappings to sequences of *conclusion-bounded* GLAV mappings is left for future work. We have also seen that there are sequences of LAV mappings for which no SO tgd is a uniform limit. Are there structural properties that characterize when a sequence of GLAV mappings has an SO tgd as a pointwise limit? In this vein, we have offered Conjecture 20. A related interesting open problem is whether schema mappings with target constraints are powerful enough to express pointwise limits or uniform limits of sequences of arbitrary GLAV schema mappings. We have some preliminary evidence that this is plausible, but much more work remains to be done.

We believe that the work reported in this paper provides a new perspective on the study of schema mappings by examining them from a dynamic viewpoint. As stated earlier, our original motivation came from schema-mapping optimization and, in particular, from the idea that "complex" schema mappings can be "approximated" by "simpler" ones. It remains to be

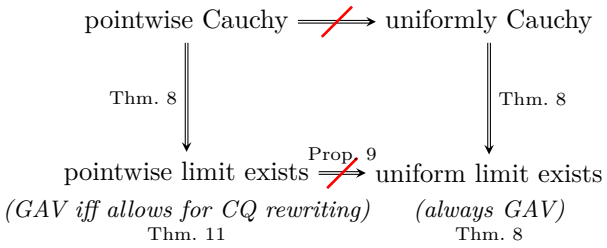


Figure 1 Overall picture for GAV schema mappings.

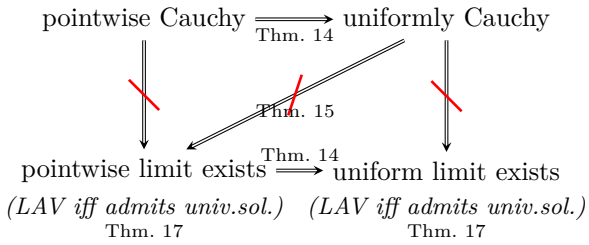


Figure 2 Overall picture for LAV schema mappings.

seen whether the work reported here will lead to applications to schema-mapping optimization. We believe, however, that the study of the limiting behavior of schema mappings via metric spaces is interesting in its own right.

We also note there are several areas in theoretical computer science where the study of limiting behavior of objects has produced results that were significant in their own right and also had fruitful consequences. For example, starting with the work of Fagin [4], there has been an extensive investigation of the asymptotic probabilities of logical properties and of 0-1 laws for various logics of interest in computer science. More recently, there has been a study of *profinite words*, which has found applications to automata theory and to the satisfiability problem for variants of monadic second-order logic (see, e.g., [17, 20]). Note that the profinite words form the completion of a metric space on words in which the distance is based on the size of the largest deterministic finite automaton needed to separate two words. Finally, as mentioned in the previous section, there is a direct connection between graph limits in the monograph [15] by Nešetřil and Ossona de Mendez and the completion of the metric space $(\mathcal{P}(\mathrm{Inst}(\mathbf{T})), d)$, which may merit further exploration. It should also be pointed out that, motivated from the study of large-scale networks, there has been an extensive body of work on a notion of graph limits arising from converging sequences of *homomorphism densities*; a detailed account of this work is given in the monograph [13] by Lovász. In addition, Nešetřil and Ossona de Mendez [16] developed a general framework for limits of graphs and relational structures; in that framework, different fragments of first-order logic are used to define different notions of limits arising from converging sequences of the frequencies that first-order formulas in the fragment at hand are satisfied by an assignment (homomorphism densities correspond to the fragment consisting of all quantifier-free conjunctive queries). Homomorphisms, metric completions, and representations of limits of finite structures play a central role in [13, 16]. The precise connections with the work reported here will have to be worked out in a future investigation.

Acknowledgments. The research of Phokion G. Kolaitis on this paper was partially supported by NSF Grant IIS-1217869. The research of Reinhard Pichler, Emanuel Salinger, and Vadim Savenkov was supported by the Austrian Science Fund, projects (FWF):P25207-N23 and (FWF):Y698, and the Vienna Science and Technology Fund, project ICT12-015. Emanuel Sallinger is currently supported by the EPSRC grant EP/M025268/1.

References

1 Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.
2 Marcelo Arenas, Jorge Pérez, Juan Reutter, and Cristian Riveros. The language of plain SO-tgds: Composition, inversion and structural properties. *Journal of Computer and System Sciences*, 79(6):763–784, 2013. doi:10.1016/j.jcss.2013.01.002.
3 Philip A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.
4 Ronald Fagin. Probabilities on finite models. *J. Symb. Log.*, 41(1):50–58, 1976.
5 Ronald Fagin and Phokion G. Kolaitis. Local transformations and conjunctive-query equivalence. In *PODS*, pages 179–190, 2012. doi:10.1145/2213556.2213583.
6 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, May 2005.
7 Ronald Fagin, Phokion G. Kolaitis, Alan Nash, and Lucian Popa. Towards a theory of schema-mapping optimization. In *PODS*, pages 33–42, 2008. doi:10.1145/1376916.1376922.
8 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.
9 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Schema mapping evolution through composition and inversion. In *Schema Matching and Mapping*, pages 191–222. Springer, 2011.
10 Ingo Feinerer, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. On the undecidability of the equivalence of second-order tuple generating dependencies. *Inf. Syst.*, 48:113–129, 2015. doi:10.1016/j.is.2014.09.003.
11 Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, pages 61–75, 2005.
12 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
13 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012.
14 Jayant Madhavan and Alon Y. Halevy. Composing mappings among data sources. In *VLDB*, pages 572–583, 2003. URL: http://www.vldb.org/conf/2003/papers/S18P01.pdf.
15 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.
16 Jaroslav Nešetřil and Patrice Ossona de Mendez. A unified approach to structural limits, and limits of graphs with bounded tree-depth. arXiv:1303.6471, 2013.
17 Jean-Eric Pin. Profinite methods in automata theory. In *STACS*, pages 31–50, 2009.
18 Satish Shirali and Harkrishan Vasudeva. *Metric spaces*. Springer, 2006.
19 Balder ten Cate and Phokion G. Kolaitis. Structural characterizations of schema-mapping languages. In *ICDT*, pages 63–72, 2009. doi:10.1145/1514894.1514903.
20 Szymon Torunczyk. Languages of profinite words and the limitedness problem. In *ICALP*, pages 377–389, 2012.

Reasoning About Integrity Constraints for Tree-structured Data*

Wojciech Czerwiński[1], Claire David[2], Filip Murlak[1], and Paweł Parys[1]

1 **University of Warsaw, Warsaw, Poland**
{wczerwin,fmurlak,parys}@mimuw.edu.pl
2 **Université Paris-Est Marne-la-Vallée, Champs-sur-Marne, France**
claire.david@u-pem.fr

Abstract

We study a class of integrity constraints for tree-structured data modelled as data trees, whose nodes have a label from a finite alphabet and store a data value from an infinite data domain. The constraints require each tuple of nodes selected by a conjunctive query (using navigational axes and labels) to satisfy a positive combination of equalities and a positive combination of inequalities over the stored data values. Such constraints are instances of the general framework of XML-to-relational constraints proposed recently by Niewerth and Schwentick. They cover some common classes of constraints, including W3C XML Schema key and unique constraints, as well as domain restrictions and denial constraints, but cannot express inclusion constraints, such as reference keys. Our main result is that consistency of such integrity constraints with respect to a given schema (modelled as a tree automaton) is decidable. An easy extension gives decidability for the entailment problem. Equivalently, we show that validity and containment of unions of conjunctive queries using navigational axes, labels, data equalities and inequalities is decidable, as long as none of the conjunctive queries uses both equalities and inequalities; without this restriction, both problems are known to be undecidable. In the context of XML data exchange, our result can be used to establish decidability for a consistency problem for XML schema mappings. All the decision procedures are doubly exponential, with matching lower bounds. The complexity may be lowered to singly exponential, when conjunctive queries are replaced by tree patterns, and the number of data comparisons is bounded.

1998 ACM Subject Classification H.2.1 [Database Management] Logical Design

Keywords and phrases data trees, integrity constraints, unions of conjunctive queries, schema mappings, entailment, containment, consistency

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.20

1 Introduction

Static analysis is an area of database theory that focuses on deciding properties of syntactic objects, like queries, integrity constraints, or data dependencies. The unifying paradigm is that since these objects are mostly user-generated, they tend to be small; hence, higher complexities are tolerable. Typical problems include satisfiability, validity, containment, and equivalence of queries [5, 8, 23, 24], and consistency and entailment of constraints [13, 26]. More specialized tasks include query rewriting in data integration scenarios [22], and manipulating schema

* The first, third, and fourth author of this paper were supported by Poland's National Science Centre grant no. UMO-2013/11/D/ST6/03075.

mappings in data exchange and schema evolution scenarios [1, 12]. Many of these problems are equivalent to satisfiability of fragments of first order logic, possibly over a restricted class of structures, but they are rarely presented this way, because the involved fragments are tailored for specific applications, and usually do not form natural sublogics. As the satisfiability problem for first order logic is undecidable even for relatively simple fragments, in static analysis undecidability is always close [15, 16].

In this paper we present a decidability result (with tight complexity bounds) for a problem in static analysis for tree-structured data. The specific data model we consider is that of data trees: finite ordered unranked trees whose nodes have a label from a finite alphabet and store a data value from an infinite data domain. The problem has three possible formulations:

- consistency modulo schema for a class of integrity constraints;
- validity modulo schema for a class of queries; and
- consistency for a class of schema mappings.

The more general problems of entailment (or implication) of constraints and containment of queries are – as is often the case – very close to their restricted counterparts listed above, and can be solved by easy modifications of our decision procedure.

Our basic setting is that of consistency of integrity constraints; it seems best suited for proofs and – in combination with entailment – the most appealing. We consider *non-mixing constraints* of the forms

$$\alpha(\bar{x}) \Rightarrow \eta_{\sim}(\bar{x}) \qquad \text{and} \qquad \alpha(\bar{x}) \Rightarrow \eta_{\nsim}(\bar{x})$$

that require each tuple $\bar{x}$ of nodes selected by α to satisfy, respectively, a positive combination of equalities $\eta_{\sim}$ or a positive combination of inequalities $\eta_{\nsim}$ over the stored data values. As tuple selectors $\alpha(\bar{x})$ we use conjunctive queries over the signature including label tests and the usual navigational axes.

What is the expressive power of non-mixing constraints? Let us first look at what they cannot do. Being first-order constraints, they cannot compare full subtrees, unlike some other formalisms [17, 18]. They have purely universal character (can be written as universal sentences of first order logic), so they cannot express general inclusion dependencies nor foreign keys, as these need quantifier alternation. Finally, the inability to mix freely data equalities and inequalities within a single constraint makes them unable to express general functional dependencies. What can they do, then?

Non-mixing integrity constraints can be seen as a special case of the general framework of XML-to-relational constraints (X2R constraints) introduced by Niewerth and Schwentick [25]. Within this framework they cover a wide subclass of functional dependencies, dubbed XKFDs, which are particularly well suited for tree-structured data and include W3C XML Schema key and unique constraints [14], as well as absolute and relative XML keys by Arenas, Fan, and Libkin [2], and XFDs by Arenas and Libkin [3]. XKFDs can be expressed with non-mixing constraints of the form $\alpha(\bar{x}) \Rightarrow \eta_{\nsim}(\bar{x})$; that is, using only data inequalities.

Constraints of the form $\alpha(\bar{x}) \Rightarrow \eta_{\sim}(\bar{x})$ – that is, using only equalities – can express all sorts of finite data domain restrictions, either to a specific set of constants or to a set of data values taken from the data tree (the latter can be seen as a limited variant of inclusion constraints), as well as cardinality restrictions over data values.

The novelty of our work is that we allow these two kinds of constraints simultaneously. Unrestricted mixing of data equalities and inequalities in constraints would immediately lead to undecidability [6], but for non-mixing constraints we can show decidability of the consistency problem, and a slight extension of the proof gives decidability for entailment (with the same complexity bounds).

Under the second interpretation our result shows decidability of validity and containment for unions of conjunctive queries where each conjunctive query can use either data equality or inequality, but never both. Seen this way, our result is a uniform extension of decidability results for UCQs using only data equality, and UCQs using only data inequality by Björklund, Martens and Schwentick [6] (see also [9]). However, it cannot be obtained via a combination of techniques used in these cases, as they are virtually contradictory: they require assuming that almost all data values in counter-examples are, respectively, different and equal. If data equalities and inequalities are mixed freely in UCQs, even validity is undecidable [6].

In its third incarnation, our result gives decidability of the consistency problem for XML schema mappings with source integrity constraints, which asks to decide if there exists a source instance which satisfies the integrity constraints and admits a target instance satisfying the requirements imposed by the schema mapping.

In all three cases the decision procedure is doubly exponential. This bound is tight, as already validity modulo schema for UCQs over trees without data values is 2ExpTime-complete [6]. We show that restricting the CQs to tree patterns does not help. However, the complexity does drop to ExpTime-complete when we replace CQs with tree patterns and bound the number of variables used in data comparisons.

The remainder of the paper begins with a precise definition of non-mixing constraints and a short discussion of their scope (Section 2). Then we present the decision procedure for consistency of non-mixing constraints (Section 3), followed by a detailed discussion of the entailment problem, the lower-complexity fragment, the relationships with existing constraint formalisms, and the two alternative interpretations of our results (Section 4). We conclude with a brief discussion of further extensions and open questions (Section 5). An appendix containing the missing proofs is available at: `http://www.mimuw.edu.pl/~fmurlak/papers/concon.pdf`.

2 Non-mixing constraints

Preliminaries

Let us fix a finite labelling alphabet Γ and a countably infinite set of data values $\mathbb{D}$. A *data tree* t is a finite ordered unranked tree whose nodes are labelled with elements of Γ by function $\mathsf{lab}_t : \mathsf{dom}_t \to \Gamma$, and with elements of $\mathbb{D}$ by function $\mathsf{val}_t : \mathsf{dom}_t \to \mathbb{D}$; here, dom_t stands for the *domain* of tree t, that is, the set of its nodes. If $\mathsf{lab}_t(v) = a$ and $\mathsf{val}_t(v) = d$, we say that node v has label a and stores data value d. For a node v of t, we write t_v for the fragment of t consisting of trees rooted at v itself and at all preceding siblings of v. By slight abuse of notation we write $t - t_v$ for the remaining part of t.

We abstract schemas as tree automata in the "previous sibling, last child" variant. A *tree automaton* $\mathcal{A}$ is a tuple (Q, q_0, F, δ), where Q is a finite set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of accepting states, and $\delta \subseteq Q \times Q \times \Gamma \times Q$ is a set of transitions. Being in a node v of the input tree t, the automaton has processed t_v. The state for node v depends on the label of v and the states from the previous sibling and the last child of v. In leftmost siblings and in leaves we resort to imaginary nodes outside of the actual tree t, which are always assigned the initial state q_0. Formally, let dom_t^{cl} be the set containing each node of t, an artificial previous sibling for each leftmost sibling in t, and an artificial (last) child for each leaf in t. A run of $\mathcal{A}$ on t is a function $\rho : \mathsf{dom}_t^{cl} \to Q$ such that $\rho(v) = q_0$ for every node $v \in \mathsf{dom}_t^{cl} - \mathsf{dom}_t$, and for every node $v \in \mathsf{dom}_t$ with previous sibling v_{ps} and last child v_{lc} there is a transition $(\rho(v_{ps}), \rho(v_{lc}), \mathsf{lab}_t(v), \rho(v)) \in \delta$. A run ρ is *accepting* if it assigns a state from F to the root of t, and a tree t is *accepted* by $\mathcal{A}$ if it admits an accepting run.

To facilitate the use of the standard first order semantics, we model data trees as relational structures over signature

$$\mathsf{sig}_{dt} = \{\downarrow, \downarrow^+, \rightarrow, \rightarrow^+, \sim, \not\sim\} \cup \Gamma \cup \mathbb{D} \cup \widehat{\mathbb{D}}$$

with $\widehat{\mathbb{D}} = \{\hat{d} \mid d \in \mathbb{D}\}$; that is, we have

- binary relations: *child* $\downarrow$, *descendant* $\downarrow^+$, *next sibling* $\rightarrow$, and *following sibling* $\rightarrow^+$;
- data equality relation $\sim$ and data inequality relation $\not\sim$ that contain pairs of nodes storing, respectively, the same data value and different data values;
- unary relation a for each label $a \in \Gamma$;
- unary relations d and $\hat{d}$ for each data value $d \in \mathbb{D}$ that contain nodes storing, respectively, data value d and some data value different from d.

Signature sig_{dt} is infinite (because of $\mathbb{D}$ and $\widehat{\mathbb{D}}$), but queries use only finite fragments.

A *conjunctive query* $\alpha(x_1, \ldots, x_n)$ over a signature sig is a first order formula of the form

$$\exists y_1 \ldots \exists y_m \, \beta(x_1, \ldots, x_n, y_1, \ldots, y_m) \,,$$

where $\beta(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a conjunction of atoms over signature sig and variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, such that each variable occurs in at least one atom.

Definition

In their most general form, *non-mixing integrity constraints* σ are formulas of the form

$$\alpha(\bar{x}) \Rightarrow \eta_\sim(\bar{x}) \wedge \eta_{\not\sim}(\bar{x})$$

where

- $\alpha(\bar{x})$ is a conjunctive query over signature $\mathsf{sig}_{nav} = \{\downarrow, \downarrow^+, \rightarrow, \rightarrow^+\} \cup \Gamma$;
- $\eta_\sim(\bar{x})$ is a positive Boolean combination of atoms over $\mathsf{sig}_\sim = \{\sim\} \cup \mathbb{D}$ and variables $\bar{x}$;
- $\eta_{\not\sim}(\bar{x})$ is a positive Boolean combination of atoms over $\mathsf{sig}_{\not\sim} = \{\not\sim\} \cup \widehat{\mathbb{D}}$ and variables $\bar{x}$.

Query α is called the *selector* of σ, and $\eta_\sim, \eta_{\not\sim}$ are its *assertions*. Non-mixing constraints have the usual semantics of first order logic formulas: a data tree t satisfies constraint σ, denoted $t \models \sigma$, if each tuple $\bar{v}$ of nodes of t selected by α satisfies both $\eta_\sim$ and $\eta_{\not\sim}$; that is,

$$t \models \alpha(\bar{v}) \quad \text{implies} \quad t \models \eta_\sim(\bar{v}) \wedge \eta_{\not\sim}(\bar{v}) \,.$$

For a set Σ of non-mixing constraints, we write $t \models \Sigma$ if $t \models \sigma$ for all $\sigma \in \Sigma$.

Note that $\alpha \Rightarrow \eta_\sim \wedge \eta_{\not\sim}$ is equivalent to $\{\alpha \Rightarrow \eta_\sim \,,\ \alpha \Rightarrow \eta_{\not\sim}\}$. Consequently, each set Σ of non-mixing constraints is equivalent to $\Sigma_\sim \cup \Sigma_{\not\sim}$, where $\Sigma_\sim$ is a set of constraints of the form $\alpha \Rightarrow \eta_\sim$, $\Sigma_{\not\sim}$ is a set of constraints of the form $\alpha \Rightarrow \eta_{\not\sim}$, and the sizes of $\Sigma_\sim$ and $\Sigma_{\not\sim}$ are bounded by the size of Σ. Thus, without loss of generality, we restrict our attention to sets of constraints of the form $\Sigma_\sim \cup \Sigma_{\not\sim}$, which do not mix $\mathsf{sig}_\sim$ and $\mathsf{sig}_{\not\sim}$ (hence "non-mixing"). One can also assume that α is quantifier free: $\exists \bar{y} \, \alpha(\bar{x}, \bar{y}) \Rightarrow \eta(\bar{x})$ is equivalent to $\alpha(\bar{x}, \bar{y}) \Rightarrow \eta(\bar{x})$.

Scope

Using non-mixing constraints one can express a variety of useful constraints. Let us consider a database storing information about banks, each in a separate sub-document. We want each bank to be identified by its BIC number. This **key constraint** can be expressed as

$$q_{\text{BIC}}(x, x') \wedge q_{\text{BIC}}(y, y') \wedge x \neq y \;\Rightarrow\; x' \not\sim y'$$

where q_{BIC} selects the root of the sub-document for bank, and the node storing the BIC number. Depending on the schema, query q_{BIC} could be for instance $q_{\text{BIC}}(x, x') = \texttt{bank}(x) \wedge x \downarrow x' \wedge \texttt{BIC}(x')$. Node inequality $\neq$ is not part of the signature, but can be expressed using sig_{nav}. Assuming that the roots of the sub-documents for banks are siblings, $x \neq y$ can be replaced by $x \rightarrow^+ y$. In general, we also need to consider four other possible ways in which two different nodes x and y can be positioned in a tree (up to swapping x and y):

$$x \downarrow^+ y\,, \quad x \rightarrow^+ z \wedge z \downarrow^+ y\,, \quad z \downarrow^+ x \wedge z \rightarrow^+ y\,, \quad \text{and} \quad z \downarrow^+ x \wedge z \rightarrow^+ z' \wedge z' \downarrow^+ y\,,$$

which means that we need five non-mixing constraints to express a single key constraint.

Another natural constraint is that account numbers should be different for every account within the same bank, but different banks may use the same account numbers. Such a **relative key constraint** can also be expressed as

$$\texttt{bank}(z) \wedge z \downarrow^+ x \wedge z \downarrow^+ y \wedge q_{\text{ACC}}(x, x') \wedge q_{\text{ACC}}(y, y') \wedge x \neq y \;\Rightarrow\; x' \nsim y'\,,$$

where $q_{\text{ACC}}(x, x')$ selects account x and its number x', similarly to q_{BIC}.

We can also express **multi-attribute keys** (i.e. keys using composite fields). For example

$$q_{\text{BIC}}(u, u') \wedge q_{\text{BIC}}(v, v') \wedge u \downarrow^+ x \wedge v \downarrow^+ y \wedge q_{\text{ACC}}(x, x') \wedge q_{\text{ACC}}(y, y') \wedge x \neq y \;\Rightarrow\; u' \nsim v' \vee x' \nsim y'$$

asserts that BIC and account number form an absolute key, not relative to bank sub-document.

If, as a result of redundancy, BIC appears in several places within a bank sub-document, using the **singleton constraint**

$$\texttt{bank}(x) \wedge x \downarrow^+ x' \wedge \texttt{BIC}(x') \wedge x \downarrow^+ x'' \wedge \texttt{BIC}(x'') \;\Rightarrow\; x' \sim x''$$

we can guarantee that each time it gives the same value (for the same bank).

Assume now that each bank has a director and several branches, each of them having a team of employees among which one is the manager of the branch. The information about each employee is stored in a sub-document of its branch's sub-document. Each employee reports either to the manager of the branch or directly to the director of the bank. Using a conjunctive query $q_{\text{SUPER}}(x, y, z)$, we can select the director's ID node x, the branch manager's ID node y and the node z storing the supervisor's ID for an employee of the same branch. The constraint on employee's supervisor can be encoded as

$$q_{\text{SUPER}}(x, y, z) \;\Rightarrow\; x \sim z \vee y \sim z\,.$$

Following this idea we can express **inclusion constraints of a restricted form**, where the intended superset is a tuple of values that can be selected by a conjunctive query. This includes **enumerative domain restrictions**, like the constraint

$$\texttt{creditCard}(x) \wedge x \downarrow^+ x' \wedge \texttt{brand}(x') \Rightarrow \mathit{Visa}(x') \vee \mathit{MasterCard}(x') \vee \mathit{AmericanExpress}(x')\,,$$

ensuring that banks issue only Visa, Master Card, and American Express cards. Unrestricted inclusion constraints are beyond the scope of our formalism. Indeed, non-mixing constraints cannot be violated by removing nodes, which is not the case even for the simplest unary inclusion constraints, like *each value stored in an a node is also stored in a b node.*

Our formalism is also capable of expressing **cardinality constraints**. Assume, for instance, that banks support charity projects by delegating their employees to help. The projects are organized by category (culture, education, environment, etc.) and each project sub-document carries the list of involved employees. For the sake of balance, we want each

category to involve at most ten different employees in total. This can be imposed by selecting eleven employee nodes below a single category node and imposing at least two of them to carry the same data value by means of a long disjunction of data equalities. We can also ensure that no employee is involved in more than three different projects: the conjunctive query selects four different project nodes and an employee for each of them; the assertion imposes at least two of the four employees to have different ID.

Let us remark that while these constraints look clumsy expressed as non-mixing constraints, one can easily imagine a syntactic-sugar layer on top of our formalism. The point is that all these constraints can be rewritten as non-mixing constraints of linear size (except for the cardinality constraints, where the numerical bounds would be typically given in decimal).

In Section 4 we examine the expressive power of non-mixing constraints further by comparing them to other existing formalisms.

3 Consistency problem

Our main result is decidability of the consistency problem for non-mixing constraints:

Problem:	**Consistency of non-mixing constraints**
Input:	A set Σ of non-mixing constraints, a tree automaton $\mathcal{A}$.
Question:	Is there a data tree such that $t \in L(\mathcal{A})$ and $t \models \Sigma$?

More precisely, we show the following theorem, establishing tight complexity bounds.

▶ **Theorem 1.** *Consistency of non-mixing constraints is* 2ExpTime*-complete.*

The proof of Theorem 1 is based on a simple idea with a geometric flavour, but does not require any specialist knowledge from geometry or linear algebra. The key fact is an upper bound on the number of affine subspaces that constitute an intersection of unions of affine subspaces of an Euclidean space; it has a short elementary proof. From this bound we infer a "bounded data cut" model property for non-mixing constraints, where by *data cut* of a data tree t, denoted by $datacut(t)$, we mean the maximum over nodes $v \in \mathsf{dom}_t$ of the number of data values shared by t_v and $t - t_v$.

A *subspace* of $\mathbb{D}^\ell$ is a subset of $\mathbb{D}^\ell$ defined by equating pairs of coordinates and fixing coordinates; that is, it is a set of points $(x_1, x_2, \ldots, x_\ell)$ in space $\mathbb{D}^\ell$ defined by a conjunction of equalities of the form $x_i = x_j$ or $x_i = d$ where $d \in \mathbb{D}$. Each nonempty subspace of $\mathbb{D}^\ell$ can be defined by a *canonical* set of at most ℓ equalities such that

- for each coordinate i we have either $x_i = x_j$ with $i < j$, or $x_i = d$ with $d \in \mathbb{D}$, or nothing;
- each coordinate j occurs at most once on the right side of an equality; and
- no data value d is used in more than one equality.

A subspace of $\mathbb{D}^\ell$ has *dimension* k if its canonical definition consists of $\ell - k$ equalities. In other words, each equality that does not follow from the others decreases the dimension by one. To enhance intuitions, let us remark that if we equip $\mathbb{D}^\ell$ with the structure of linear space by assuming that $\mathbb{D}$ is a field, this notion of dimension coincides with the classical notion of dimension for affine subspaces (of which the subspaces above are a special case).

An intersection $X \cap Y$ of subspaces X, Y is also a subspace, defined by the conjunction of conditions defining X and Y. If $X \not\subseteq Y$, then the canonical definition of $X \cap Y$ contains at least one more equation, consequently, the dimension of $X \cap Y$ is strictly smaller than the dimension of X. Similarly, intersecting unions of subspaces, we obtain a union of subspaces; the following lemma gives a bound on the size of such union.

▶ **Lemma 2.** *Let $Z_1, Z_2, \ldots, Z_m \subseteq \mathbb{D}^\ell$ be such that each Z_i is a union of at most n subspaces of $\mathbb{D}^\ell$. Then, $Z_1 \cap Z_2 \cap \cdots \cap Z_m$ is a union of at most n^ℓ subspaces of $\mathbb{D}^\ell$.*

Proof. Assume that $Z_1 \cap Z_2 \cap \cdots \cap Z_{i-1}$ is a union $X_1 \cup X_2 \cup \cdots \cup X_p$ of subspaces of $\mathbb{D}^\ell$. We can write Z_i as $Y_1 \cup Y_2 \cup \cdots \cup Y_n$, where some of subspaces Y_k may be empty. We have

$$Z_1 \cap Z_2 \cap \cdots \cap Z_i = (X_1 \cup X_2 \cup \cdots \cup X_p) \cap Z_i = (X_1 \cap Z_i) \cup (X_2 \cap Z_i) \cup \cdots \cup (X_p \cap Z_i)\,.$$

Let us examine a single $X_j \cap Z_i$. If $X_j \subseteq Y_k$ for some k, then $X_j \cap Z_i = X_j$. Otherwise, $X_j \cap Z_i$ is a union of n subspaces, $X_j \cap Y_1, X_j \cap Y_2, \ldots, X_j \cap Y_n$, where each $X_j \cap Y_k$ is either empty or has dimension strictly smaller than X_j. Thus, when $X_1 \cup X_2 \cup \cdots \cup X_p$ is intersected with Z_i, each X_j either does not change, or falls apart into at most n subspaces of strictly smaller dimension; if X_j is a point, in the second possibility it disappears.

Now, consider the following process: begin with $\mathbb{D}^\ell$, a single subspace of dimension ℓ, and then intersect with Z_i for i from 1 to m, one by one. It follows immediately from the observation above that we cannot obtain more than n^ℓ subspaces in this process. ◀

We remark that the bound in Lemma 2 is tight, as shown by the following example.

▶ **Example 3.** [1] Assume $0, 1 \in \mathbb{D}$ and let $Z_i = \left\{\bar{x} \subseteq \mathbb{D}^\ell \;\middle|\; x_i = 0 \vee x_i = 1\right\}$ for $i = 1, 2, \ldots, \ell$. Then $Z_1 \cap Z_2 \cap \cdots \cap Z_\ell = \{0,1\}^\ell$ is a union of 2^ℓ (disjoint) subspaces of $\mathbb{D}^\ell$ of dimension 0.

Based on this geometric fact, in Lemma 5 we bound the data cut of data trees witnessing consistency of non-mixing constraints. The proof relies on a simple compositionality property for conjunctive queries over trees, stated in Lemma 4 (see Appendix A for proof).

▶ **Lemma 4.** *Let $\alpha(\bar{x}, \bar{y})$ be a conjunction of atoms over sig_{nav}, where $\bar{x}$ and $\bar{y}$ are disjoint, and let w be a node of a data tree t. For all tuples $\bar{u}, \bar{u}'$ of nodes from t_w and tuples $\bar{v}, \bar{v}'$ of nodes from $t - t_w$, if $t \models \alpha(\bar{u}, \bar{v})$ and $t \models \alpha(\bar{u}', \bar{v}')$, then $t \models \alpha(\bar{u}, \bar{v}')$ and $t \models \alpha(\bar{u}', \bar{v})$.*

▶ **Lemma 5.** *If $\Sigma_\sim \cup \Sigma_{\not\sim}$ is satisfied in a data tree t, it is also satisfied in some data tree t' obtained from t by changing data values, such that $\mathit{datacut}(t') \leq \ell \cdot 2^\ell \cdot (\ell + m)^{\ell^2} \cdot |\Sigma_\sim|$, where ℓ, m are the maximal numbers of variables and predicates from $\mathbb{D} \cup \hat{\mathbb{D}}$ in constraints of $\Sigma_\sim$.*

Proof. Assume that $t \models \Sigma_\sim \cup \Sigma_{\not\sim}$ and let w be a node of the data tree t. We shall replace all but $\ell \cdot 2^\ell \cdot (\ell + m)^{\ell^2} \cdot |\Sigma_\sim|$ data values used in t_w with distinct fresh data values, thus ensuring that in the resulting tree the number of data values used both in t_w and $t - t_w$ is bounded by $\ell \cdot 2^\ell \cdot (\ell + m)^{\ell^2} \cdot |\Sigma_\sim|$. As the fresh data values are be distinct, the new $\sim$ relation over nodes of t is a subset of the old one. In consequence, the operation does not increase the number of data values shared by $t_{w'}$ and $t - t_{w'}$ for other nodes w'. For the same reason, the obtained tree still satisfies $\Sigma_{\not\sim}$. We only need to ensure that $\Sigma_\sim$ is not violated.

Consider a constraint $\alpha \Rightarrow \eta_\sim$ in $\Sigma_\sim$. Recall that we assume that α is quantifier free. Let $\bar{x}$, $\bar{y}$ be a partition of variables used in α (one of the tuples $\bar{x}$, $\bar{y}$ may be empty). We shall indicate the partition of variables by writing the constraint as $\alpha(\bar{x}, \bar{y}) \Rightarrow \eta_\sim(\bar{x}, \bar{y})$. Directly from the definition it follows that $t \models \alpha \Rightarrow \eta_\sim$, if and only if for each partition $\bar{x}, \bar{y}$ of variables in α, for each tuple $\bar{u}$ of nodes from t_w and each tuple $\bar{v}$ of nodes from $t - t_w$, if $t \models \alpha(\bar{u}, \bar{v})$, then $t \models \eta_\sim(\bar{u}, \bar{v})$.

Fix a partition $\bar{x}$, $\bar{y}$. By Lemma 4, this is equivalent to: for all tuples $\bar{u}, \bar{u}'$ of nodes from t_w and all tuples $\bar{v}, \bar{v}'$ of nodes from $t - t_w$, if $t \models \alpha(\bar{u}, \bar{v})$ and $t \models \alpha(\bar{u}', \bar{v}')$, then $t \models \eta_\sim(\bar{u}, \bar{v}')$.

[1] Provided by Michał Pilipczuk, during the Warsaw Automata Group's research camp *Autobóz 2015*.

Let us turn this into a condition on stored data values. Define $\eta(\bar{x},\bar{y})$ as the formula obtained from $\eta_\sim(\bar{x},\bar{y})$ by replacing $\sim$ with $=$, and $d(x)$ with $x=d$ for all variables x and all $d\in\mathbb{D}$. Reformulating the condition above we obtain: for each tuple $\bar{u}$ of nodes from t_w such that $t\models\alpha(\bar{u},\bar{v})$ for some tuple $\bar{v}$ of nodes from $t-t_w$, the tuple $\mathsf{val}_t(\bar{u})$ of data values belongs to the set

$$Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})}=\bigcap_{\bar{v}'}\left\{\bar{c}\in\mathbb{D}^{|\bar{x}|}\;\middle|\;\eta(\bar{c},\mathsf{val}_t(\bar{v}'))\right\},$$

where $\bar{v}'$ ranges over tuples of nodes from $t-t_w$ satisfying $t\models\alpha(\bar{u}',\bar{v}')$ for some tuple $\bar{u}'$ of nodes from t_w.

Let $\mathsf{val}':\mathsf{dom}_t\to\mathbb{D}$ be a new data labelling of t. Since we are only planning to change data values stored in nodes of t_w, the labelling val' does not violate $\Sigma_\sim$ if and only if for each constraint $\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})$ in $\Sigma_\sim$ (with each partition of variables), for each tuple $\bar{u}$ of nodes from t_w that satisfies $t\models\alpha(\bar{u},\bar{v})$ for some tuple $\bar{v}$ of nodes from $t-t_w$, the tuple $\mathsf{val}'(\bar{u})$ belongs to the set $Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{z})}$. Writing $\eta(\bar{x},\mathsf{val}_t(\bar{v}'))$ in the disjunctive normal form, we see that the set $\left\{\bar{c}\in\mathbb{D}^{|\bar{x}|}\;\middle|\;\eta(\bar{c},\mathsf{val}_t(\bar{v}'))\right\}$ is a union of subspaces of $\mathbb{D}^{|\bar{x}|}$. How many subspaces? The canonical definition of each nonempty subspace has for each coordinate i either an equality $x_i=x_j$ for some $j>i$, or an equality $x_i=d$ for some $d\in\mathbb{D}$, or nothing. In our case, d is a data value used explicitly in η or occurring in the data tuple $\mathsf{val}_t(\bar{v}')$. Consequently, the number of these subspaces can be bounded by $(N+|\bar{x}|+|\bar{y}|)^{|\bar{x}|}$, where N is the number of data values used explicitly in η. That is, $Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})}$ is an intersection of unions of at most $(N+|\bar{x}|+|\bar{y}|)^{|\bar{x}|}$ subspaces of $\mathbb{D}^{|\bar{x}|}$. By Lemma 2, it is a union of at most $(N+|\bar{x}|+|\bar{y}|)^{|\bar{x}|^2}$ subspaces. Let us analyse the restrictions it puts on val'. We have declared that we shall only replace some data values in t_w with fresh data values. That is, equalities of the form $x_i=x_j$ in the definitions of those subspaces will not get violated. It remains to check that equalities of the form $x_i=d$ are not violated. Each subspace involves at most $|\bar{x}|$ such equalities, so for set $Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})}$ we have at most $|\bar{x}|\cdot(N+|\bar{x}|+|\bar{y}|)^{|\bar{x}|^2}$ of them. Let $D\subseteq\mathbb{D}$ be the set of data values occurring in these equalities for all sets $Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})}$, with $\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})$ ranging over constraints from $\Sigma_\sim$ with all possible partitions of variables. We have $|D|\leq|\Sigma_\sim|\cdot 2^\ell\cdot\left(\ell\cdot(m+\ell)^{\ell^2}\right)$, where ℓ and m are the maximal numbers of variables and predicates from $\mathbb{D}\cup\hat{\mathbb{D}}$ in constraints from $\Sigma_\sim$. Altogether, a labelling val' that replaces each data value from $\mathbb{D}-D$ used in t_w with a fresh data value does not violate $\Sigma_\sim$. ◀

Having obtained Lemma 5, we are ready to prove Theorem 1. In the proof we use register tree automata. These are tree automata, equipped additionally with a finite number of registers. The registers can store data values (from $\mathbb{D}$) read from the data tree or just guessed, and compare them with data values seen later in the tree. Formally, a *register tree automaton* $\mathcal{A}$ is a tuple (Q,q_0,F,k,δ), where Q is a finite set of states, $q_0\in Q$ is an initial state, $F\subseteq Q$ is a set of accepting states, k is the number of registers, and δ is a set of transitions. A transition in δ is of the form $(q_{ps},q_{lc},a,q_{cur},E)$, where $q_{ps},q_{lc},q_{cur}\in Q$, $a\in\Gamma$, and E is an equivalence relation over $\{val\}\cup\big(\{ps,lc,cur\}\times\{1,\ldots,k\}\big)$ in which each equivalence class contains at most one element from $\{l\}\times\{1,\ldots,k\}$ for each $l\in\{ps,lc,cur\}$.

Next, we define how $\mathcal{A}$ runs on a data tree t. Similarly to a standard tree automaton, to compute the state and the register valuation in some node, automaton $\mathcal{A}$ uses the state and the register valuation from the previous sibling and the last child of that node. The equivalence relation E of the transition specifies which data values are equal: val denotes the data value stored in the tree node, (ps,i) denotes the data value assigned to the i-th register in the previous sibling, and similarly for (lc,i) and (cur,i). We do not specify initial register valuations: the register valuation in an imaginary previous sibling or last child may

be arbitrary. Formally, a run $\rho = (\rho_Q, \rho_{\mathbb{D}})$ of $\mathcal{A}$ on t is given by functions $\rho_Q : \mathsf{dom}_t^{cl} \to Q$ and $\rho_{\mathbb{D}} : \mathsf{dom}_t^{cl} \times \{1, \ldots, k\} \to \mathbb{D}$ such that for every node $v \in \mathsf{dom}_t^{cl} - \mathsf{dom}_t$ it holds that $\rho_Q(v) = q_0$, and for every node $v \in \mathsf{dom}_t$ with previous sibling v_{ps} and last child v_{lc} there is a transition $(\rho_q(v_{ps}), \rho_q(v_{lc}), \mathsf{lab}_t(v), \rho_q(v), E) \in \delta$ with $E = \{(a, b) \mid as(a) = as(b)\}$ where $as(val) = \mathsf{val}_t(v)$, $as(ps, i) = \rho_{\mathbb{D}}(v_{ps}, i)$, $as(lc, i) = \rho_{\mathbb{D}}(v_{lc}, i)$, $as(cur, i) = \rho_{\mathbb{D}}(v, i)$. Run ρ is accepting if ρ_Q assigns a state from F to the root of t.

Our automata cannot store the same value in more than one register, like in the seminal paper by Kaminski and Francez [18]; this guarantees polynomial-time emptiness test.

▶ **Theorem 6.** *It is decidable in polynomial time whether a given register tree automaton accepts at least one data tree.*

Kaminski and Francez [19] introduced register automata for words; the tree variant was proposed by Kaminski and Tan [20]. The emptiness problem was already considered in [20], but since our automaton model is slightly different, we give a sketch of the proof in Appendix B.

To prove Theorem 1, we reduce consistency of non-mixing constraints to emptiness of register tree automata of doubly exponential size. The register automaton we construct computes a representation of tuples selected from the input data tree by the selector queries of $\Sigma_\sim \cup \Sigma_{\not\sim}$. To explain how this can be done, we need some auxiliary notions.

A *partial valuation* of variables $x_1, \ldots, x_k$ is a function

$$f \colon \{x_1, x_2, \ldots, x_k\} \to \mathsf{dom}_t \cup \{\bot\}\,.$$

If $f(x_i) \neq \bot$, we say that x_i is *matched* at $f(x_i)$, and if $u_i = \bot$ we say that x_i is *not matched*. Two partial valuations of the same set of variables are *disjoint*, if no variable is matched by both of them. The *union* of disjoint partial valuations f, g of variables $x_1, \ldots, x_k$ is given as

$$(f \cup g)(x_i) = \begin{cases} f(x_i) & \text{if } f(x_i) \neq \bot \\ g(x_i) & \text{otherwise} \end{cases}$$

A *partial matching* of $\alpha(\bar{x})$ in t_w is a partial valuation f of variables $\bar{x}$ such that variables are matched only in the nodes of t_w, each atom in $\alpha(f(\bar{x}))$ that does not contain $\bot$ holds true, and each atom that contains both a node from t_w and $\bot$ is of the form

$$w \to \bot, \quad w' \to^+ \bot, \quad \bot \downarrow w', \quad \text{or} \quad \bot \downarrow^+ v\,,$$

where w' is a preceding sibling of w or w itself, and v is an arbitrary node of t_w. The last condition means that each such atom can be made true (independently of others) by replacing $\bot$ with a node from $t - t_w$, unless w has no following siblings or no ancestors in t.

If $t \models \alpha(\bar{u})$, each partial valuation matching a subset of variables x_i at nodes u_i from t_w is a partial matching of α. Conversely, if a partial matching f matches all variables $\bar{x}$, then $t \models \alpha(f(\bar{x}))$. Note, however, that not every partial matching can be extended so that it matches all variables: remaining atoms may be satisfiable on their own, but not together.

Proof of Theorem 1. Let $\Sigma_\sim \cup \Sigma_{\not\sim}$ be a set of non-mixing integrity constraints and let $\mathcal{A}$ be a tree automaton. We shall construct a register tree automaton $\mathcal{B}$ such that $\mathcal{B}$ accepts at least one data tree if and only if some tree accepted by $\mathcal{A}$ satisfies $\Sigma_\sim \cup \Sigma_{\not\sim}$. Automaton $\mathcal{B}$ will be the product of automaton $\mathcal{A}$ and a register tree automaton $\mathcal{C}$ of doubly exponential size. We would like automaton $\mathcal{C}$ to check if the input data tree satisfies $\Sigma_\sim \cup \Sigma_{\not\sim}$, but this does not seem possible. Instead, automaton $\mathcal{C}$ will accept data trees that satisfy $\Sigma_\sim \cup \Sigma_{\not\sim}$ up to a relabelling of data values. This is sufficient to guarantee correctness of the reduction.

For all tuples $\bar{u}$ such that $t \models \alpha(\bar{u})$, automaton $\mathcal{C}$ should check that the corresponding assertion $\eta(\bar{u})$ holds. For that it suffices to know the equalities between data values stored in nodes $\bar{u}$ and the ones used explicitly in η. The information about these equalities is collected node by node: when automaton $\mathcal{C}$ is in a node w of input data tree t, it has processed t_w, and $t - t_w$ remains to be processed. To compute these equalities we need to keep track of data values used in t_w that will also occur in $t - t_w$. We use registers to store them. More precisely, automaton $\mathcal{C}$ stores in registers M data values used explicitly in $\Sigma_\sim \cup \Sigma_{\not\sim}$ (this is the only way it can perform comparisons with them), and some data values shared by t_w and $t - t_w$. Number M is bounded by the size of the input. The number of values shared by t_w and $t - t_w$ is in general unbounded, but for a tree of optimal data cut we can store all of them, as, by Lemma 5, their number is bounded by some N, singly exponential in the size of the input. These two sets of values may overlap, but $M + N$ registers are enough to store them. Automaton $\mathcal{C}$ cannot determine which data values are used in $\Sigma_\sim \cup \Sigma_{\not\sim}$, nor which data values seen in t_w will be used again in $t - t_w$, so it uses nondeterminism to fill in the registers. The first M registers are guessed in each initial configuration and never changed; the remaining ones are updated (nondeterministically) in each processed node w.

In states, automaton $\mathcal{C}$ remembers for each selector $\alpha(\bar{x})$ a subset $\Delta_{\alpha(\bar{x})}$ of

$$\{\mathsf{r}_1, \mathsf{r}_2, \ldots, \mathsf{r}_{M+N}, \top_1, \top_2, \ldots, \top_\ell, \bot\}^{|\bar{x}|},$$

where ℓ is the maximal number of variables used in constraints in $\Sigma_\sim \cup \Sigma_{\not\sim}$. Each such tuple represents a partial matching of $\alpha(\bar{x})$ in t_w, and the whole $\Delta_{\alpha(\bar{x})}$ represents a set of such partial matchings. The intended meaning of the symbolic values is as follows: r_i in coordinate j of the tuple means that variable x_j is matched and its data value is stored in the i-th register, $\top_i$ means that variable x_j is matched but we do not store the corresponding data value and consider it different from all others, $\bot$ means that variable x_j has not been matched. In the initial state, each $\Delta_{\alpha(\bar{x})}$ is $\{(\bot, \bot, \ldots, \bot)\}$. When some $\Delta_{\alpha(\bar{x})}$ contains a tuple that does not use $\bot$ and does not satisfy the corresponding assertion $\eta(\bar{x})$, automaton $\mathcal{C}$ rejects the input tree t immediately.

Let us describe the transition relation. Assume that automaton $\mathcal{C}$ is about to determine the state in a node w. Let w' and w'' be, respectively, the previous sibling and the last child of w. The set of partial matchings of $\alpha(\bar{x})$ in t_w depends only on the sets of partial matchings in $t_{w'}$ and $t_{w''}$, and the label of w. Indeed, a partial valuation of $\bar{x}$ is a partial matching of $\alpha(\bar{x})$ in t_w if it is the union of disjoint partial matchings of $\alpha(\bar{x})$ in $t_{w'}$ and $t_{w''}$ extended by matching some (yet unmatched) variables at node w, respecting two conditions. For all atoms $x_i \to x_j$, $x_i \to^+ x_j$ in $\alpha(\bar{x})$, either x_i, x_j are both matched in $t_{w''}$ or none is; and the new matching of variables at w does not violate the definition of partial matching. The latter can be expressed as follows:

- if $\alpha(\bar{x})$ contains $x_i \downarrow x_j$ or $x_i \downarrow^+ x_j$, we may match x_i at w only if x_j is matched in $t_{w''}$; for $x_i \downarrow x_j$, if x_j is matched, we must match x_i, unless it is matched already;
- if $\alpha(\bar{x})$ contains $x_i \to x_j$ or $x_i \to^+ x_j$, we may match x_j at w only if x_i is matched in $t_{w'}$; for $x_i \to x_j$, if x_i is matched, we must match x_j, unless it is matched already;
- if $\alpha(\bar{x})$ contains $a(x_i)$, we may match x_i at w only if $\mathsf{lab}_t(w) = a$.

Checking the conditions above requires only information about which variables are matched in $t_{w'}$ and $t_{w''}$; the used tree nodes are not relevant. Consequently, one can determine the set of tuples representing partial matchings in t_w based on the sets of tuples representing partial matchings in $t_{w'}$ and $t_{w''}$, and the label of the current node w. In the resulting representation we use r'_i and r''_i for the copies of the registers from the previous sibling w' and the last child w'', and curr as a symbolic representation of the data value in the current node w. Next, we

update the content of the registers. Registers $1, 2, \ldots, M$ must store the same values in both branches of the run, and they are never modified. Registers $M+1, M+2, \ldots, M+N$ are filled with distinct values chosen nondeterministically. Some of these values may be copied from registers $M+1, M+2, \ldots, M+N$ in one of the two branches of the run, or from the current node w. After updating the content of registers, we must reflect the changes in sets $\Delta_{\alpha(\bar{x})}$. If the new value in the i-th register is equal to the one in the j-th register in node w' for some i, j, all occurrences of r'_j are replaced by r_i; analogously for node w''. Similarly, if the value in the i-th register is equal to $\mathsf{val}_t(w)$, we replace curr with r_i. At this point, all remaining symbols r'_j, r''_k, and curr represent data values that are not expected to be seen again. For each such symbol, in each tuple separately, we replace all its occurrences with a distinct unused element of set $\{\top_1, \top_2, \ldots, \top_\ell\}$; there are always enough elements available.

Let us see that automaton $\mathcal{C}$ is correct. By construction, it accepts each data tree satisfying $\Sigma_\sim \cup \Sigma_{\not\sim}$, whose data cut is at most N: initially it guesses correctly the M data values used in $\Sigma_\sim \cup \Sigma_{\not\sim}$, and then in each processed node w it guess a content for the remaining N registers such that all data value shared by t_w and $t - t_w$ are stored. Lemma 5 guarantees that if $\Sigma_\sim \cup \Sigma_{\not\sim}$ is satisfiable, it is satisfiable in a data tree of data cut at most N.

It remains to see that if $\mathcal{C}$ accepts a data tree t, then by modifying data values in t we can obtain a data tree satisfying $\Sigma_\sim \cup \Sigma_{\not\sim}$. Observe first that each data tree accepted by $\mathcal{C}$ satisfies $\Sigma_\sim$, modulo a relabelling of data values (there is no guarantee that automaton $\mathcal{C}$ correctly guesses the values used in $\Sigma_\sim \cup \Sigma_{\not\sim}$, but since registers always store different values, each permutation of $\mathbb{D}$ that sends the content of the i-th register to the i-th data value used in $\Sigma_\sim \cup \Sigma_{\not\sim}$ fixes it). Indeed, in each partial matching represented by a given tuple, actual data values represented in the tuple by the same symbol are always equal. So, when an assertion $\eta_\sim$ holds for the representing tuple, it also holds for each represented matching. The same is not ensured for assertions $\eta_{\not\sim}$. Indeed, when automaton $\mathcal{C}$ sees a data value that is not in the registers, it assumes that it is different from all data values seen before and now represented by $\top_1, \top_2, \ldots, \top_\ell$. But in fact, this value might have been seen, and later forgotten and replaced with a value $\top_i$ in the tuples representing partial matchings. Consequently, in some represented matchings, the same data value may be represented with two (or more) different symbols. We shall show how to modify the data values in t to obtain a data tree, also accepted by automaton $\mathcal{C}$, for which data values represented with different symbols are always different.

Let ρ be an accepting run of automaton $\mathcal{C}$ on data tree t. For each node w and each data value d that is not stored in the registers after processing node w, replace all occurrences of d in t_w and in the corresponding part ρ_w of run ρ with a fresh data value. The resulting run ρ' is an accepting run of automaton $\mathcal{C}$ on the resulting tree t'. In data tree t', two nodes store the same data value if and only if they stored the same value in t and on the shortest path connecting these nodes this data value was always kept in a register in run ρ. Applying a permutation of $\mathbb{D}$ if necessary, we may assume that the values stored in registers $1, 2, \ldots, M$ are indeed the data values used in $\Sigma_\sim \cup \Sigma_{\not\sim}$. Hence, as we have argued, $t' \models \Sigma_\sim$ because automaton $\mathcal{C}$ accepts t'. Let us check that $t' \models \Sigma_{\not\sim}$. If a data value $\mathsf{val}_t(w)$, represented symbolically by curr, is already stored in some register r_i during run ρ, then by definition of register tree automaton it is identified with r_i. Symbol curr is not identified with any r_i only if $\mathsf{val}_t(w)$ is stored in none of the registers. By construction of t', $\mathsf{val}_{t'}(w)$ does not occur in t'_w, and it is correct to represent it with a distinct symbolic value (which is also done in run ρ'). Thus, in each partial matching represented by a given tuple, data values represented by different symbols are indeed different, and if assertion $\eta_{\not\sim}$ holds for the representing tuple, it also holds for each represented matching. This concludes the proof that $t' \models \Sigma_\sim \cup \Sigma_{\not\sim}$.

Since number $M + N$ is singly exponential, so is the number of tuples representing partial matchings. Each state of automaton $\mathcal{C}$ stores a set of such tuples for each constraint, so $\mathcal{C}$ is doubly exponential; it can be also constructed in doubly exponential time. By Theorem 6, we test emptiness of the product of tree automaton $\mathcal{A}$ and register tree automaton $\mathcal{C}$ in time polynomial in the size of the product. This amounts to a 2ExpTime algorithm.

The matching lower bound can be shown via a reduction from the acceptance problem for alternating Turing machines using exponential space, already for constraints that use tree patterns as selectors and assertions over $\mathsf{sig}_{\sim}$ only; see Appendix C for details. ◀

4 Extensions, connections, and applications

Entailment of non-mixing constraints

A static analysis problem more general than consistency is entailment. Recall that a set of constraints Σ' is *entailed* by a set of constraints Σ modulo a tree automaton $\mathcal{A}$, written as $\Sigma \models_{\mathcal{A}} \Sigma'$, if for each data tree t accepted by automaton $\mathcal{A}$,

$$t \models \Sigma \quad \text{implies} \quad t \models \Sigma' .$$

The entailment problem is then defined as follows:

Problem:	**Entailment problem for non-mixing constraints**
Input:	sets Σ, Σ' of non-mixing constraints, tree automaton $\mathcal{A}$
Question:	$\Sigma \models_{\mathcal{A}} \Sigma'$?

Entailment is a more general problem than consistency, but for non-mixing constraints the results on consistency generalize to entailment almost effortlessly.

▶ **Theorem 7.** *Entailment of non-mixing constraints is* 2ExpTime*-complete.*

Proof. Inconsistency is a special case of entailment: Σ is inconsistent with respect to an automaton $\mathcal{A}$ if and only if $\Sigma \models_{\mathcal{A}} \bot$, where $\bot$ is an inconsistent set of constraints, say $\{a(x) \Rightarrow 0(x) \wedge 1(x) \mid a \in \Gamma\}$. Thus, the lower bound follows.

Lemma 5 shows that witnesses for consistency can have bounded data cut. The same is true for counter-examples to entailment. Suppose $t \models \Sigma$ and $t \not\models \Sigma'$. Then, $t \models \alpha'(\bar{u}) \wedge \neg\eta'(\bar{u})$ for some constraint $\alpha'(\bar{x}) \Rightarrow \eta'(\bar{x})$ from Σ' and some tuple $\bar{u}$ of nodes of t. Let D_0 be the set of data values used in the nodes $\bar{u}$. We can repeat the construction of t' word for word, except that we replace the set D of values not to be touched by $D \cup D_0$. This increases $\mathit{datacut}(t')$ by the maximal number of variables in the constraints of Σ'.

The automata construction in the proof of Theorem 1 is modified similarly. It is enough to consider Σ' consisting of a single constraint $\alpha'(\bar{x}) \Rightarrow \eta'(\bar{x})$. Let M' be the number of data values used explicitly in Σ'. The number of registers is increased by $M' + |\bar{x}|$, to accommodate these data values, and the ones stored in the nodes $\bar{u}$ that witness satisfaction of $\alpha'(\bar{x}) \wedge \neg\eta'(\bar{x})$ in t. All these values are guessed by the automaton in the beginning of the run, and never modified. In the states the automaton additionally stores a set $\Delta_{\alpha'(\bar{x})}$ of tuples representing partial matchings of $\alpha'(\bar{x})$ such that the matched nodes store the initially guessed values (symbolic values $\top_i$ are not used in $\Delta_{\alpha'(\bar{x})}$). The automaton behaves like before, additionally checking that $\Delta_{\alpha'(\bar{x})}$ contains a tuple without $\bot$ that satisfies $\neg\eta'(\bar{x})$.

The argument does not change if $\eta'(\bar{x})$ mixes predicates from $\mathsf{sig}_{\sim}$ and $\mathsf{sig}_{\nsim}$. ◀

A singly exponential fragment

A closer look at the complexity of our algorithm reveals that it is double exponential only in the maximal number ℓ of variables in the constraints, which appears in two roles: in the exponent of the bound on the number of data values stored in registers, and as the length of tuples representing partial matchings of selectors. A slightly more detailed analysis of the proof of Lemma 5 shows that in the first role ℓ could be replaced by the maximal number ℓ' of variables used in the assertions. Indeed, since data equalities involve only variables used in assertions, everything is in fact happening in a space of dimension at most ℓ'. While limiting the size of selector queries to lower complexity makes little sense, limiting the number of variables in assertions seems acceptable. But what about the second role of ℓ?

The need to represent all partial matchings (up to data equality type) comes from the fact that the automaton is evaluating conjunctive queries all over the tree. The standard technique to lower complexity in such cases is to replace conjunctive queries with tree patterns, which are essentially tree-structured conjunctive queries. In the most basic form, with only $\downarrow$ and $\downarrow^+$ axes allowed, a *tree pattern* is a conjunctive query π over signature $\{\downarrow, \downarrow^+\} \cup \Gamma$, such that graph $(A_\pi, \downarrow_\pi \cup \downarrow_\pi^+)$ is a directed tree, where $\mathbb{A}_\pi = (A_\pi, \downarrow_\pi, \downarrow_\pi^+, \{a_\pi\}_{a\in\Gamma})$ is the canonical relational structure associated to query π in the usual way: the universe A_π is the set of variables of π, and relations are given by the respective atoms in π. For non-mixing integrity constraints, restricting selectors to tree patterns does not suffice to lower the complexity: the reduction in Appendix C uses only such constraints (and no assertions over $\mathsf{sig}_{\nsim}$). But together with the bound on the number of variables in assertions – it does (see Appendix D).

▶ **Proposition 8.** *For non-mixing constraints whose selectors are tree patterns and whose assertions use constantly many variables, consistency and entailment are* ExpTime*-complete.*

Static analysis of unions of conjunctive queries

Our results can be reinterpreted in the framework of static analysis of unions of conjunctive queries (UCQs). Note that $t \not\models \alpha(\bar{x}) \Rightarrow \eta(\bar{x})$ if and only if $t \models \exists \bar{x}\, \alpha(\bar{x}) \wedge \neg\eta(\bar{x})$. It follows immediately that the problem of validity of UCQs over signature sig_{dt} that never mix predicates from $\mathsf{sig}_\sim$ and $\mathsf{sig}_{\nsim}$ – call them *non-mixing UCQs* – reduces in polynomial time to *inconsistency* of non-mixing constraints. Similarly, containment of such queries reduces to entailment of non-mixing constraints. The converse reduction is also possible, but it involves exponential blow-up, caused by rewriting arbitrary Boolean combinations in disjunctive normal form. This correspondence brings our results very close to the work by Björklund, Martens, and Schwentick on static analysis for UCQs over signature $\mathsf{sig}_{nav} \cup \{\sim, \nsim\}$ [6].

On one hand, our results immediately give the following decidability result for the setting considered by Björklund, Martens, and Schwentick (constraints used in our 2ExpTime lower bound can be rewritten without blow-up).

▶ **Theorem 9.** *Over* $\mathsf{sig}_{nav} \cup \{\sim, \nsim\}$*, both validity of non-mixing UCQs and containment of UCQs in non-mixing UCQs (with respect to a given tree automaton) are* 2ExpTime*-complete.*

Results of Björklund, Martens, and Schwentick give 2ExpTime upper bound for containment in UCQs over $\mathsf{sig}_{nav} \cup \{\sim\}$ and UCQs over $\mathsf{sig}_{nav} \cup \{\nsim\}$. The original work is on CQs, but arguments for UCQs are the same [9]. Essentially, they amount to an observation that in counter-examples to containment $p \subseteq q$, all data values can be set equal (in the case with $\nsim$) or different (in the case with $\sim$), except for a bounded number of them needed to witness satisfaction of p; such counter-examples can be easily encoded as trees over a finite alphabet, and recognized by an automaton evaluating p and q in the usual way. Theorem 9 extends

both these results. Since we have both $\sim$ and $\nsim$ in query q, we cannot assume that all data values are equal, nor that all are different; our more involved approach seems necessary.

The third relevant result of [6] is that containment of CQ p over $\mathsf{sig}_{nav} \cup \{\sim\}$ in CQ q over $\mathsf{sig}_{nav} \cup \{\sim, \nsim\}$ is 2ExpTime-complete. It looks stronger than ours because query q can mix $\sim$ and $\nsim$. In fact, it is much weaker, depending entirely on the fact that q is a single CQ, not a UCQ. More specifically, the argument is as follows: if q uses $\nsim$, the answer is *yes* if and only if p is not satisfiable with respect to the tree automaton (a witness can have all data values equal, so it definitely does not satisfy q); if q does not use $\sim$, we are back in the case of UCQs over $\mathsf{sig}_{nav} \cup \{\sim\}$.

On the other hand, some results of Björklund, Martens, and Schwentick give a broader context to our results. They show that validity with respect to a given automaton is already 2ExpTime-complete for unions of conjunctive queries over signature sig_{nav}, that is, for trees without data. Consequently, restricting only assertions of non-mixing constraints would not lower the complexity. This is complementary to our 2ExpTime lower bound, which shows hardness for constraints using tree patterns as selectors. Hence, the only way to lower the complexity is to restrict both, selectors and assertions. Björklund, Martens, and Schwentick also show that for UCQs over $\mathsf{sig}_{nav} \cup \{\sim, \nsim\}$ validity is undecidable; this means that we cannot go beyond non-mixing assertions.

XML constraints

Non-mixing constraints form an instance of the general framework of XML-to-relational (X2R) constraints proposed by Niewerth and Schwentick [25]: tuple selectors are conjunctive queries over sig_{nav}, schemas are tree automata, and relational constraints are positive quantifier-free formulas over $\mathsf{sig}_{\sim}$ or $\mathsf{sig}_{\nsim}$. Niewerth and Schwentick investigate two classes of relational constraints: functional dependencies (FDs) and XML-key FDs (XKFDs). In the X2R setting, tuple selectors return nodes and data values, in separate "columns". In an FD

$$A_1\, A_2\, \ldots\, A_m \to B\,,$$

$A_1, A_2, \ldots, A_m, B$ are arbitrary columns (referring either to nodes or to data values); in an XKFD, B is required to be a node column. Our setting captures XKFDs, but not general FDs. Consider an X2R constraint given by a CQ $\alpha(x_1, \ldots, x_n)$ populating a table with tuples $(x_1, \ldots, x_n, @x_1, \ldots, @x_n)$, where $@x_i$ stands for the data value stored in the node represented by variable x_i, and an XKFD $x_1, \ldots, x_j, @x_{j+1}, \ldots, @x_{n-1} \to x_n$ (it makes no sense to use both x_i and $@x_i$ in the same constraint). Such constraint can be rewritten as

$$\alpha(x_1, \ldots, x_n) \wedge \alpha(x_1, \ldots, x_j, x'_{j+1}, \ldots, x'_n) \wedge x_n \neq x'_n \;\Rightarrow\; x_{j+1} \nsim x'_{j+1} \vee \cdots \vee x_{n-1} \nsim x'_{n-1}\,,$$

which can be turned into a set of five non-mixing constraints by replacing $x_n \neq x'_n$ with simple subqueries describing possible ways of arranging two different nodes in a tree, as explained in Section 2. Note that these constraints do not use $\sim$. Hence, for XKFDs with UCQs over sig_{nav} as tuple selectors decidability of entailment follows already from the results on containment of UCQs over $\mathsf{sig}_{nav} \cup \{\sim\}$, discussed in the previous subsection; the challenge tackled by Niewerth and Schwentick is to determine the exact complexity and identify tractable fragments.

If we replace the XKFD above with an FD $x_1, \ldots, x_j, @x_{j+1}, \ldots, @x_{n-1} \to @x_n$ we have

$$\alpha(x_1, \ldots, x_n) \wedge \alpha(x_1, \ldots, x_j, x'_{j+1}, \ldots, x'_n) \;\Rightarrow\; x_{j+1} \nsim x'_{j+1} \vee \cdots \vee x_{n-1} \nsim x'_{n-1} \vee x_n \sim x'_n\,,$$

which cannot be expressed without mixing $\sim$ and $\nsim$. As we have explained, consistency and entailment is undecidable for such constraints, but one can investigate fragments with restricted schemas and tuple-selectors. This is what Niewerth and Schwentick do.

As XKFDs with tree patterns as tuple-selectors can express XML Schema key and unique constraints [14], XML keys by Arenas, Fan, and Libkin [2], and XFDs by Arenas and Libkin [3], so can non-mixing constraints. A technical subtlety is that some of these classes of constraints apply to nodes of a specified type (playing the role of a state in XML Schemas). As proposed by Niewerth and Schwentick, we can deal with it by annotating tree nodes with types (verified by the automaton encoding the schema), and let the patterns refer to types and labels. This slight extension does not affect our complexity bounds. Also, XML Schema key constraints demand that each field path selects at most one node, and XML Schema key constraints demand exactly one node; this can be checked by the automaton too. In practice, one often wants at most (or exactly) one *data value*, not tree node. This may or may not be equivalent. To express that at most one data value is selected, we can use the singleton constraints discussed in Section 2. Note that this requires assertions over $\mathsf{sig}_{\sim}$.

Consistency of XML schema mappings

Schema mappings are a formalism used in data exchange scenarios to specify relations between instances of two database schemas, a *source* schema and a *target* schema [1, 11, 21]. In the basic setting for XML [4], schemas can be abstracted as tree automata, and the relation between source and target instances can be defined by a set Σ of dependencies of the form

$$\alpha(\bar{x}) \Rightarrow \alpha'(\bar{x})$$

where α, α' are conjunctive queries over sig_{nav}, treated as queries selecting data values, not nodes. That is, a pair of data trees (t, t') satisfies dependency σ of the form above, written as $(t, t') \models \sigma$, if

$$\left\{\mathsf{val}_t(\bar{u}) \;\middle|\; t \models \alpha(\bar{u})\right\} \subseteq \left\{\mathsf{val}_{t'}(\bar{u}) \;\middle|\; t' \models \alpha'(\bar{u})\right\}.$$

The *consistency problem for XML schema mappings* [4] is to decide for a given schema mapping $\mathcal{M} = (\mathcal{A}, \mathcal{A}', \Sigma)$, whether there exists a tree t accepted by automaton $\mathcal{A}$ and a tree t' accepted by automaton $\mathcal{A}'$ such that $(t, t') \models \Sigma$. This problem is known to be decidable: without loss of generality one may assume that all data values in t and t' are equal, and use standard automata techniques ignoring data values. This is not only uninspiring theoretically, but also not very practical: an instance with all data values equal is not a convincing witness that the mapping makes sense. What if the source schema includes constraints, say XML Schema key or unique constraints? We cannot assume that all data values are equal any more. As we have argued in the previous subsection, such constraints can be expressed with non-mixing constraints, which leads us to the problem of *consistency with source constraints*, a common generalization of consistency of constraints and schema mappings: given a schema mapping $\mathcal{M} = (\mathcal{A}, \mathcal{A}', \Sigma)$ and a set of non-mixing constraints Σ_{src}, decide if there exist a tree t accepted by automaton $\mathcal{A}$ and a tree t' accepted by automaton $\mathcal{A}'$ such that $t \models \Sigma_{src}$ and $(t, t') \models \Sigma$.

The following lemma gives the connection between XML schema mappings and non-mixing constraints that allows us to apply our decidability result. It was proved in a slightly different but equivalent form in [10]. A non-mixing constraint with *free data value predicates* uses additional unary predicate symbols in the assertions. A data tree t satisfies a set Σ

of such constraints (possibly sharing some additional predicate symbols) if it satisfies Σ' obtained from Σ by replacing each additional predicate symbol with some $d \in \mathbb{D}$. Free data value predicates are not problematic for the consistency algorithm, as it can guess the data values to replace them; up to equality type with respect to data values already used in Σ, there are only exponentially many possibilities.

▶ **Lemma 10.** *For each schema mapping $\mathcal{M} = (\mathcal{A}, \mathcal{A}', \Sigma)$ one can compute in doubly exponential time sets $\Sigma^1_\sim, \Sigma^2_\sim, \ldots, \Sigma^m_\sim$ of non-mixing constraints with free data value predicates, each obtained from Σ by replacing target-side queries $\alpha'(\bar{x})$ with assertions $\eta_\sim(\bar{x})$ of exponential size, such that for each data tree t, $t \models \Sigma_i$ for some $1 \leq i \leq m$ if and only $(t, t') \models \Sigma$ for some data tree t' accepted by automaton $\mathcal{A}'$.*

Thus, mapping $\mathcal{M}$ is consistent with source constraints Σ_{src} if and only if at least one of the sets $\Sigma^i_\sim \cup \Sigma_{src}$ obtained via Lemma 10 is consistent with respect to automaton $\mathcal{A}$. Since the number of variables in each involved constraint is linear, the latter can be tested in 2ExpTime, as the algorithm from Section 3 is doubly exponential only in the maximal number of variables. As Lemma 10 translates mappings into constraints with assertions over $\mathsf{sig}_\sim$, even if Σ_{src} is just a set of key constraints (expressible with assertions over $\mathsf{sig}_{\not\sim}$), we need the full power of non-mixing constraints, allowing assertions over $\mathsf{sig}_\sim$ and $\mathsf{sig}_{\not\sim}$.

5 Conclusions

We have shown that consistency and entailment of non-mixing constraints are decidable. Both problems are 2ExpTime-complete, but become ExpTime-complete when we restrict selector queries to tree patterns and bound the number of variables in assertions. We have reinterpreted these results in terms of validity and containment of conjunctive queries, as well as consistency of schema mappings. The latter setting best illustrates the benefits of combining assertions over $\mathsf{sig}_\sim$ and $\mathsf{sig}_{\not\sim}$. Indeed, equalities are involved even in the simplest schema mappings, and inequalities allow to cover key constraints over the source database.

We worked with ordered trees, but our results immediately carry over to unordered trees: as long as the signature does not contain the horizontal axes, one can freely move back and forth between ordered and unordered trees by forgetting the sibling order or introducing it arbitrarily. As the 2ExpTime lower bound does not use the horizontal axes, it holds also for unordered trees. The reduction can be adapted to the case of unlabelled trees: one can simulate labels with unique small tree gadgets attached to the main nodes of the tree and use the automaton to ensure that each main node has exactly one gadget attached. However, referring to the gadgets with selector queries requires either the next sibling or the following sibling relation. For unordered unlabelled trees the complexity might drop.

Our decidability results can be pushed further to constraints with selector queries defined in monadic second order logic (MSO) over the signature sig_{nav} (an extension of first order logic with second-order quantification over sets). To this end, one reproves Lemma 5 using compositionality of MSO instead of Lemma 4; this requires considering MSO types and makes the bound on the data cut non-elementary. One concludes by using decidability of MSO over the signature $\mathsf{sig}_{nav} \cup \mathsf{sig}_\sim \cup \mathsf{sig}_{\not\sim}$ on trees of bounded data cut [7].

Yet another context in which our results can be applied is the static analysis of XPath queries [5]. In their basic form, our results immediately give decidability of the containment problem (in the presence of a schema) for unions of XPath queries without negation, where each query uses either equality or inequality, but never both. The extension to MSO discussed above allows free use of negation, as long as equalities and inequalities are not

used under negation. But it does not give satisfying complexity bounds. It seems worthwhile to investigate the complexity issue deeper, and look at ways of extending this towards full XPath, for which decidability of satisfiability remains open.

Acknowledgements. We thank the anonymous referees for their insightful questions.

References

1 Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

2 Marcelo Arenas, Wenfei Fan, and Leonid Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.*, 38(3):841–880, 2008. doi:10.1137/050646895.

3 Marcelo Arenas and Leonid Libkin. A normal form for XML documents. *ACM Trans. Database Syst.*, 29:195–232, 2004. doi:10.1145/974750.974757.

4 Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2), 2008. doi:10.1145/1346330.1346332.

5 Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008. doi:10.1145/1346330.1346333.

6 Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing conjunctive queries over trees using schema information. In *Proc. MFCS 2008*, pages 132–143, 2008. doi:10.1007/978-3-540-85238-4_10.

7 Mikołaj Bojańczyk, Filip Murlak, and Adam Witkowski. Containment of monadic datalog programs via bounded clique-width. In *Proc. ICALP 2015*, pages 427–439, 2015. doi:10.1007/978-3-662-47666-6_34.

8 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. STOC 1977*, pages 77–90, 1977. doi:10.1145/800105.803397.

9 Claire David, Amélie Gheerbrant, Leonid Libkin, and Wim Martens. Containment of pattern-based queries over data trees. In *Proc. ICDT 2013*, pages 201–212, 2013. doi:10.1145/2448496.2448521.

10 Claire David, Piotr Hofman, Filip Murlak, and Michał Pilipczuk. Synthesizing transformations from XML schema mappings. In *Proc. ICDT 2014*, pages 61–71, 2014. doi:10.5441/002/icdt.2014.10.

11 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.

12 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005. doi:10.1145/1114244.1114249.

13 Ronald Fagin and Moshe Y. Vardi. The theory of data dependencies – a survey. In *Mathematics of Information Processing*, volume 34 of *Proceedings of Symposia in Applied Mathematics*, pages 19–71, Providence, Rhode Island, 1986. American Mathematical Society.

14 S. Gao, C. M. Sperberg-McQueen, H.S. Thompson, N. Mendelsohn, D. Beech, and M. Maloney. W3C XML Schema Definition Language (XSD) 1.1, Part 1: Structures. Technical report, World Wide Web Consortium, April 2009. URL: http://www.w3.org/TR/2009/CR-xmlschema11-1-20090430/.

15 Tomasz Gogacz and Jerzy Marcinkowski. All-instances termination of chase is undecidable. In *Proc. ICALP 2014*, pages 293–304, 2014. doi:10.1007/978-3-662-43951-7_25.

16 Tomasz Gogacz and Jerzy Marcinkowski. The hunt for a red spider: Conjunctive query determinacy is undecidable. In *Proc. LICS 2015*, pages 281–292, 2015. doi:10.1109/LICS.2015.35.

17 Sven Hartmann and Sebastian Link. More functional dependencies for XML. In *Proc. ADBIS 2003*, pages 355–369, 2003. doi:10.1007/978-3-540-39403-7_27.

18 Sven Hartmann, Sebastian Link, and Thu Trinh. Solving the implication problem for XML functional dependencies with properties. In *Proc. WoLLIC 2010*, pages 161–175, 2010. doi:10.1007/978-3-642-13824-9_14.

19 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

20 Michael Kaminski and Tony Tan. Tree automata over infinite alphabets. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 386–423. Springer, 2008. doi:10.1007/978-3-540-78127-1_21.

21 Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proc. PODS 2005*, pages 61–75, 2005. doi:10.1145/1065167.1065176.

22 Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS 2002*, pages 233–246, 2002. doi:10.1145/543613.543644.

23 Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004. doi:10.1145/962446.962448.

24 Frank Neven and Thomas Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Log. Meth. Comput. Sci.*, 2(3), 2006. doi:10.2168/LMCS-2(3:1)2006.

25 Matthias Niewerth and Thomas Schwentick. Reasoning about XML constraints based on XML-to-relational mappings. In *Proc. ICDT 2014*, pages 72–83, 2014. doi:10.5441/002/icdt.2014.11.

26 Moshe Y. Vardi. Fundamentals of dependency theory. In E. Borger, editor, *Trends in Theoretical Computer Science*, pages 171–224. Computer Science Press, 1987.

Complexity of Repair Checking and Consistent Query Answering

Sebastian Arming[1], Reinhard Pichler[2], and Emanuel Sallinger[3]

1 **University of Salzburg, Salzburg, Austria**
2 **TU Wien, Vienna, Austria**
3 **University of Oxford, Oxford, UK**

Abstract

Inconsistent databases (i.e., databases violating some given set of integrity constraints) may arise in many applications such as, for instance, data integration. Hence, the handling of inconsistent data has evolved as an active field of research. In this paper, we consider two fundamental problems in this context: Repair Checking (RC) and Consistent Query Answering (CQA).

So far, these problems have been mainly studied from the point of view of data complexity (where all parts of the input except for the database are considered as fixed). While for some kinds of integrity constraints, also combined complexity (where all parts of the input are allowed to vary) has been considered, for several other kinds of integrity constraints, combined complexity has been left unexplored. Moreover, a more detailed analysis (keeping other parts of the input fixed – e.g., the constraints only) is completely missing.

The goal of our work is a thorough analysis of the complexity of the RC and CQA problems. Our contribution is a complete picture of the complexity of these problems for a wide range of integrity constraints. Our analysis thus allows us to get a better understanding of the true sources of complexity.

1998 ACM Subject Classification H.2.0 Database Management – General

Keywords and phrases inconsistency, consistent query answering, complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.21

1 Introduction

Database management systems (DBMS) allow the definition of several forms of integrity constraints (ICs) to specify restrictions on the data to be stored. The DBMS ensures that the stored data indeed satisfies the ICs. However, in modern applications where data is integrated from several sources, violations of the ICs may arise even if the data in each single source satisfies the ICs. Hence, the handling of *inconsistent* data (i.e., data violating the given ICs) has evolved as an active field of research, see e.g., [5, 6, 9, 23] for surveys and [11, 13, 16, 18, 20] for recent work. The foundations of this research were laid by Arenas et al. in [4], where the key concepts of *repairs* and of *consistent answers* were introduced.

Given a set C of ICs and a (presumably inconsistent) database instance D, a *repair* of D w.r.t. C is a database instance I which satisfies C and which *differs minimally* from D. Difference and minimality can be defined in several ways. We follow the approach of [4] where repairs are obtained from the original database by the insertion and deletion of tuples and minimality means that the symmetric set difference $\Delta(D, I)$ is minimal w.r.t. subset inclusion. More formally, let $\Delta(D, I) = (D \setminus I) \cup (I \setminus D)$. Then I is a repair of D w.r.t. C if I satisfies C and there does not exist an instance I' that satisfies C and $\Delta(D, I') \subsetneq \Delta(D, I)$.

The idea of *consistent query answers* is that even from an inconsistent database instance D, we can derive consistent information, namely those answers to a query that one would obtain over every repair I of D. More precisely, the set of consistent answers to a query Q for a given database D and ICs C is defined as $\bigcap\{Q(I) \mid I \text{ is a repair of } D \text{ w.r.t. } C\}$.

▶ **Example 1.** Consider the set of constraints $C = \{\mathsf{Course}(p, c) \to \mathsf{Prof}(p)\}$ consisting of a single inclusion dependency which states that every course must be taught by a professor. Now let us define our database $D = \{\mathsf{Course}(\mathsf{db}, \mathsf{alice}), \mathsf{Course}(\mathsf{dm}, \mathsf{bob}), \mathsf{Prof}(\mathsf{alice})\}$ that describes two courses: The course database systems (db) taught by Alice (alice) and discrete mathematics (dm) taught by Bob (bob). We observe that D is inconsistent w.r.t. C: While for the course offered by Alice there exists a corresponding tuple $\mathsf{Prof}(\mathsf{alice})$, there is no such tuple for Bob, violating the single constraint contained in C.

There are two possible *repairs*, that is, consistent databases that differ minimally from D: We can either add Bob as a professor, yielding $I_1 = D \cup \{\mathsf{Prof}(\mathsf{bob})\}$ or we remove the discrete mathematics course, yielding $I_2 = D \setminus \{\mathsf{Course}(\mathsf{dm}, \mathsf{bob})\}$. It is easy to see that I_1 and I_2 are consistent. Yet to check that these are indeed repairs, we have to show that they *differ minimally* from D in terms of symmetric difference and subset minimality. Indeed, both have just a single tuple in their symmetric difference with D, thus there can be no other consistent database instance with a smaller symmetric difference to D. In contrast, the instance $I_3 = \emptyset$ is not a repair as, while it fulfills all constraints, repair I_2 has a smaller – in terms of set inclusion – symmetric difference (I_2 removes only the offending tuple, while I_3 removes all tuples).

Let us now proceed to answering queries. Assume that we pose the query $Q = \mathsf{Course}(p, c)$. The only *consistent answer* to Q given C and D is the tuple $\mathsf{Course}(\mathsf{db}, \mathsf{alice})$ as it is contained in all repairs (both in I_1 and in I_2). In contrast, $\mathsf{Course}(\mathsf{dm}, \mathsf{bob})$ is not a consistent answer, as it is not contained in the repair I_2.

The following decision problems are crucial to deal with inconsistent data:

Repair Checking (RC)
Instance: Databases D and I and a set of constraints C
Question: Is I a repair of D w.r.t. C?

Consistent Query Answering (CQA)
Instance: A database D, a set of constraints C, and a Boolean query Q
Question: Is Q true in all repairs of D w.r.t. C?

Goal. Most research on the RC and CQA problems has focused on data complexity. For the CQA problem, this means that constraints C and query Q are considered as fixed and only the database D is allowed to vary. There are a few exceptions, as some works also consider the combined complexity, where all three parts of the input are allowed to vary, e.g. [8, 3]. However, for several kinds of integrity constraints, the combined complexity has been left unexplored. Moreover, a more detailed analysis (keeping other parts of the input fixed) is completely missing. For instance, what happens if we just fix the integrity constraints C, which is a relatively typical setting in a system where data and queries vary, but constraints stay the same? What about other types of complexity – after all, with three parts of the input, there is a total of seven reasonable combinations.

The goal of our work is a thorough analysis of the complexity of the CQA and RC problems. As an important special case, we also consider the complexity of these problems

when the arity of the relation symbols is bounded by a constant. Known results in the area provide important parts of the picture (in particular with respect to data complexity). Yet when considering all possible types of complexity (i.e., parts of the input to be fixed or varying), it turns out that for most cases the exact complexity is actually not known. In this work, we complete the picture, allowing us to get a better understanding of the true sources of complexity.

As far as the queries Q are concerned, we concentrate on the fundamental class of Boolean conjunctive queries. It can be easily verified that all of our results carry over to arbitrary conjunctive queries (i.e., asking if a given tuple is contained in the answer to Q over every repair I) and unions of conjunctive queries. We consider a number of different languages from which the constraints C are taken. The languages considered here range from first-order logic as the most powerful one to inclusion dependencies and key dependencies which are among the least expressive ones. In total, the contribution of this work is a complete picture of the complexity of the RC and CQA problems for a wide range of constraint classes.

Organization. In Section 2 (preliminaries), we introduce the constraint classes and complexity classes that are considered in this work. This will allow us to give an overview of our results in Section 3 – starting with the CQA problem (Section 3.1) and continuing with the RC problem (Section 3.2). The intuition of the results and selected proofs are given in Section 4 for repair checking and after that, in Section 5, for consistent query answering. We give concluding remarks in Section 6.

2 Preliminaries

We assume familiarity with the relational data model [1]. Below we recall some basic notions to fix the notation. A *schema* $\mathcal{S}$ is a triple $(\mathcal{U}, \mathcal{R}, \mathcal{B})$ where $\mathcal{U}$ is a countable domain, $\mathcal{R}$ is a finite set of relation symbols (each with some arity), and $\mathcal{B}$ is a finite set of built-in predicates, e.g. $\mathcal{B} = \{\leq, =\}$. Each built-in predicate from $\mathcal{B}$ comes with some fixed, not necessarily finite, relation over $\mathcal{U}$. We restrict ourselves here to the equality predicate, i.e., $\mathcal{B} = \{=\}$. However, it is easy to see that allowing other standard comparisons does not change our results. The relation symbols of $\mathcal{R}$ and $\mathcal{B}$ are called the *vocabulary* of the schema, and give rise to a language of first-order predicate logic. The *arity* of the schema is the maximum of the arities of the symbols in the vocabulary. A *database instance* is a finite set of *facts* of the form $R(a_1, \ldots, a_n)$ where $R \in \mathcal{R}$ is a relational symbol of arity n, and $a_1, \ldots, a_n$ are all elements from $\mathcal{U}$. Each database instance corresponds to a structure of the vocabulary.

We have already defined the RC and CQA problems in the introduction. We now give additional notation that will be helpful in the sequel. Given a database instance D and a first-order sentence φ such that φ is true in D, i.e. it is true in the structure corresponding to D, we write $D \models \varphi$ and say that D is *consistent* with φ. We often extend this notation to finite sets of first-order sentences Φ, writing $D \models \Phi$ if $D \models \varphi$ for every $\varphi \in \Phi$.

For an arbitrary database instance D, we define the partial order $\leq_D$ on database instances as $I \leq_D I'$ iff $\Delta(D, I) \subseteq \Delta(D, I')$. Given a set of first-order sentences C and a database instance I, we can thus say that I is a *repair* of D w.r.t. C iff $I \models C$ and there is no I' with $I' \lneq_D I$ and $I' \models C$. A *Boolean query* Q is a first-order sentence. We write $D \models_C Q$ to mean that Q is true in all repairs of D w.r.t. C. In this work, we restrict ourselves to Boolean conjunctive queries.

IC languages. Figure 1 shows the hierarchy of the IC languages considered here.

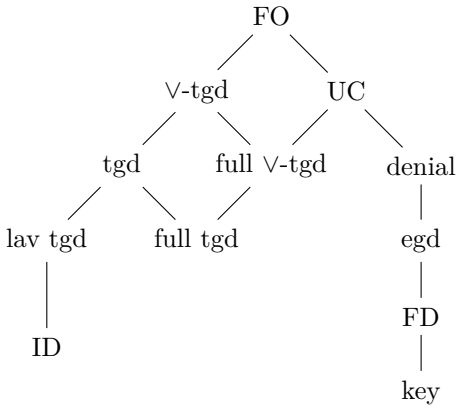


Figure 1 Hierarchy of IC languages.

Besides *domain independent first-order (FO)* sentences, all studied languages arise from restrictions on formulas of the following form

$$\forall \vec{x}(\varphi(\vec{x}) \wedge \beta(\vec{x}) \rightarrow \bigvee_{i=1}^{n} \exists \vec{y_i}\, \psi_i(\vec{x}, \vec{y_i}))$$

where φ, ψ_i are conjunctions of database atoms and β is a quantifier-free formula using only built-in predicates (i.e. equality and – in case of negated form – inequality, in this work) . To ensure safety, we require that every variable in $\vec{x}$ must occur in some relational atom in φ. We assume that constraints do not contain constants.

We call such a constraint a *full* or a *universal constraint (UC)* if it contains no existential quantifiers. A *disjunctive tuple generating dependency (∨-tgd)* has empty β, while an ordinary *tuple generating dependency (tgd)* additionally has $n = 1$. A *local-as-view (lav) tgd* is a tgd where φ is a single atom, and an *inclusion dependency (ID)* is a lav tgd where also ψ_1 is a single atom. A *denial constraint* is of the form $\forall \vec{x} \neg(\varphi(\vec{x}) \wedge \beta(\vec{x}))$ and can be thought of as a universal constraint with empty right hand side. An *equality generating dependency (egd)* is of the form $\forall \vec{x}(\varphi(\vec{x}) \rightarrow x_i = x_j)$ and can be thought of as a denial constraint where β is a single inequality. Given a relation symbol R of arity n, two sets $I \subseteq \{1, \ldots, n\}$, $J \subseteq \{1, \ldots, n\}$ and pairwise distinct variables $x_1, \ldots, x_n, y_1, \ldots, y_n$, a *functional dependency (FD)* over R is a formula of the form

$$\forall \vec{x}(R(x_1, \ldots, x_n) \wedge R(y_1, \ldots, y_n) \wedge \bigwedge_{\alpha \in I} x_\alpha = y_\alpha \rightarrow \bigwedge_{\beta \in J} x_\beta = y_\beta.$$

W.l.o.g., we may exclude trivial FDs by assuming that $I \cap J = \emptyset$ holds. Clearly, FDs are a special case of egds, since we can of course propagate the equalities on the left-hand side into the R-atoms and we can get rid of the conjunction on the right-hand side by splitting such a formula into $|J|$ formulas with identical left-hand side and a single equality on the right-hand side. A *key constraint* is an FD where $I \cup J = \{1, \ldots, n\}$ holds.

Note that in this work, we do not distinguish between individual constraints and sets of constraints. In particular, as argued above, an FD or a key constraint corresponds to a set of egds. This has to be kept in mind for the inclusions shown in Figure 1.

Complexity classes. Apart from the more familiar complexity classes P, NP, PSPACE and EXP we will refer to the following classes. First recall that Σ_2P is the class of problems

decidable by an NP Turing machine with an NP oracle, and Σ_3P contains the problems decidable by an NP Turing machine with a Σ_2P oracle. A typical complete problem for Σ_3P is the problem $\exists$QSAT$_3$, which asks whether a given quantified boolean formula with three alternating blocks of quantifiers, starting with an existential quantifier, is satisfiable. The co-classes of Σ_2P and Σ_3P are called Π_2P and Π_3P, respectively. In a similar way, Π_2EXP denotes the class of problems that can be decided by a coNEXP Turing machine with an NP oracle. The class Θ_2P consists of the problems that can be decided by a P Turing machine with nonadaptive calls to an NP oracle (i.e., calls not depending on previous calls).

Next, the Boolean hierarchy BH is the class of languages that can be expressed as a Boolean combination of languages in NP, i.e., every language in BH can be built from finitely many NP languages using intersection and complementation. An alternative characterization of BH (which is then called QH) is as the languages decidable by a P Turing machine using a constant number of calls to an NP oracle.

We use the notation $\mathcal{C}_1 \wedge \mathcal{C}_2$ to denote the conjunctive Boolean combination of two complexity classes (a language from $\mathcal{C}_1 \wedge \mathcal{C}_2$ is the intersection of a language from $\mathcal{C}_1$ and a language from $\mathcal{C}_2$). Using this notation, we define the last two relevant complexity classes, namely DP which is NP $\wedge$ coNP, and its analogue one step higher in the polynomial hierarchy DP$_2$ which is Σ_2P $\wedge$ Π_2P. For further details on the complexity classes recalled above, we refer the reader to [14, 19].

3 Overview of Results

In this section, we give an overview of our results. We start in Section 3.1 with the Consistent Query Answering (CQA) problem, as it is more natural to explain the different types of complexity here. We then continue in Section 3.2 with the Repair Checking (RC) problem.

3.1 Consistent Query Answering

In this section, we discuss the complexity of consistent query answering. Recall that the CQA problem as defined in Section 1 has three parts of input: the database instance, the set of constraints, and the query. Most of the previous research has focused on a particular variant of this problem, where constraints and query are considered fixed, and the input consists only of the database instance. The complexity of this variant is called the *data complexity*.

Yet, as we have three parts of input, there are seven possible variants to consider (as fixing all parts yields a trivial problem). In this section, we study all of the variants of CQA for all the dependency classes introduced in Section 2 and for boolean conjunctive queries as the query language.

In order to unambiguously identify the variants, we now introduce a notation based on [3, 6]. We will write CQA(X) to refer to the variant of consistent query answering where X is fixed. For example, data complexity is denoted as CQA(C,Q), since in this variant, C and Q are fixed. For the variant where no parts of the input are fixed, we simply write CQA rather than CQA(). Another, orthogonal, restriction considered in the literature is bounding the arity of the schema, which we will also consider in this work.

Overview of results. Table 1 shows the complexity of all CQA variants for each of the IC languages from Figure 1. All results in the table are completeness results, with the exception that we typically do not further classify problems in P and undecidable problems. Only for the CQA variants with FDs we show L-membership in order to match the previously known L-membership result for the data complexity of the RC problem for FDs.

Table 1 Complexity of CQA. All entries denote completeness results (except for those inside P and undecidable problems). Black triangles indicate upper (▼) and lower (▲) bounds shown in this paper. White triangles indicate previously known bounds (concrete references can be found in the paragraph on "Known results").

IC $\mathcal{L}$	CQA	CQA(D)	CQA(C)	CQA(Q)	CQA(C,Q)	CQA(D,C)	CQA(D,Q)
FO	undec	undec	undec	undec	undec	undec	undec
$\vee$-tgd	undec	undec	undec	undec	undec	undec	undec
tgd	undec	undec	undec	undec	undec △	undec ▲	undec △
lav tgd	NP ▽	NP	NP	NP	in P ▽	NP	NP ▲
ID	NP	NP	NP	in P ▼	in P	NP △	in P
UC	Π_2EXP ▼	Π_2EXP	Π_2P ▽	Π_2EXP	Π_2P	NP ▼	Π_2EXP
full $\vee$-tgd	Π_2EXP	Π_2EXP	Π_2P	Π_2EXP	Π_2P △	NP	Π_2EXP ▲
full tgd	EXP ▼	EXP	Π_2P ▲	EXP	coNP △▽	NP △	EXP ▲
denial	Π_2P ▽	BH ▼	Π_2P	Π_2P	coNP ▽	NP	BH
egd	Π_2P	BH	Π_2P	Π_2P ▲	coNP	NP	BH ▲
FD	Π_2P	NP ▼	Π_2P	coNP ▽	coNP	NP	in L ▼
key	Π_2P	NP	Π_2P ▲	coNP	coNP △	NP △	in L

Table 2 Complexity of CQA with bounded arity. Complexity for combinations not shown are as in Table 1.

IC $\mathcal{L}$	CQA	CQA(D)	CQA(Q)	CQA(D,Q)
UC	Π_3P ▼	Π_3P	Π_3P	Π_3P
full $\vee$-tgd	Π_3P	Π_3P	Π_3P	Π_3P ▲
full tgd	Π_2P ▼	Θ_2P ▼	Π_2P ▲	Θ_2P ▲

For showing the results claimed in Table 1, it is not necessary to separately show membership and hardness for each single cell. Figure 1 shows the rich inclusion structure between the constraint languages (e.g., every ID is a lav tgd). Recall that lower bounds propagate from more restricted problems to more general ones and upper bounds propagate from more general problems to more restricted ones. Thus, it suffices to show only certain membership and hardness results.

We use triangles in Table 1 to indicate the upper (▽/▼) and lower (△/▲) bounds that have to be shown in order to obtain all results given in the table. Black triangles indicate upper bounds (▼) and lower bounds (▲) shown in this paper. White triangles indicate upper bounds (▽) and lower bounds (△) given in previous work.

▶ **Theorem 2.** *For all variants of the CQA problem studied here, the complexity is as depicted in Table 1.*

We also consider CQA with bounded arity. In most cases, the complexity remains the same as for the unbounded case. Yet, where the unbounded case has provably exponential complexity, the bounded case yields complexity results inside the polynomial hierarchy. Table 2 depicts those cases.

▶ **Theorem 3.** *For the variants of the CQA problem with bounded arity, the complexity is as depicted in Table 2; in all other cases, the complexity is as given in Table 1.*

Known results. In Section 5, we will give the intuition of our results for CQA (black triangles in Tables 1 and 2). In the remainder of this section, we summarize results given in

or implicit in previous work (white triangles in Table 1). We proceed from the first to the last column of Table 1, top to bottom.

We first consider CQA (the first column of Table 1). NP-membership for lav tgds follows from [22, Theorem 4.7] as a by-product of the P-membership proof for the data complexity. Π_2P-membership for denial constraints follows from [8], where Π_2P-completeness for key constraints is stated, and it can be checked that it still holds for denial constraints.

Turning to CQA(C) (the third column of Table 1), we have that Π_2P-membership for UCs is implicit in [21, Lemma 4]. For CQA(Q) (the fourth column of Table 1), coNP-membership for FDs is established by a straightforward algorithm that guesses a counter-example and the fact that model-checking for FDs is in L.

We now proceed to CQA(C,Q) (i.e., data complexity, the fifth column of Table 1). Undecidability for tgds is shown in [22, Theorem 7.2] while P-membership for lav tgds is shown in [22, Theorem 4.7]. Π_2P-completeness for UCs is given in [21, Theorem 6]. A slight modification of the proof shows that hardness already holds for full $\vee$-tgds. We now proceed to full tgds. coNP-hardness for full tgds is given in [22, Theorem 5.5]. coNP-membership for full tgds and denial constraints is given in [21]. The matching coNP-hardness for key dependencies is shown in [10, Theorem 3.3].

We next turn to CQA(D,C) (i.e., query complexity, the penultimate column of Table 1). NP-hardness for IDs, full tgds and key dependencies follows trivially from the NP-hardness of conjunctive query answering. The result for IDs was also implicit in [8]. Finally, we proceed to CQA(D,Q) (the last column of Table 1). Undecidability for tgds follows by a slight modification of [22, Theorem 7.2].

3.2 Repair Checking

In this section, we discuss the complexity of repair checking. Recall that the RC problem as defined in Section 1 has three parts of input: two database instances and a set of constraints. As with consistent query answering, previous research has focused on data complexity, which in this case means that the two database instances are the input, while the constraints are considered as fixed.

We use the same notation as for CQA. That is, we write RC(X) to refer to the variant of repair checking where X is fixed. For example, data complexity is denoted as RC(C). For the variant where no part of the input is fixed, we again write RC rather than RC().

While the CQA problem has three dimensions (data, constraints and query), the RC problem only has two dimensions (data and constraints - as both D and I refer to data). For this reason, it is natural to treat both database instances D and I in the same way, i.e. we either fix both or none of them. In fact, it can be shown that fixing only one of the database instances does not lead to any change of complexity. That is, for the considered constraint languages, the problem variants RC(D) and RC(I) have the same complexity as RC. Thus, for every hardness result of RC, we can actually show two hardness results, namely one for RC(D) and one for RC(I). Note however, that we do not explicitly consider RC(D,C) and RC(I,C). Of course, the membership results for RC(C) implicitly carry over to the RC(D,C) and RC(I,C) cases.

Overview of results. Table 3 shows the complexity of the considered RC variants for each of the IC languages from Figure 1. The notation used is as described in Section 3.1 for Tables 1 and 2. Again all results in the table are completeness results, with the exception that we typically do not further classify problems in P. However, in addition to the cases of FDs, we now also establish an L-membership result for IDs in case of the RC(D,I) problem.

Table 3 Complexity of RC. Black triangles indicate upper (▼) and lower (▲) bounds shown in this paper. White triangles indicate previously known bounds (concrete references can be found in the paragraph on "Known results").

IC $\mathcal{L}$	RC		RC(D,I)		RC(C)	
FO	PSPACE	▼	PSPACE	△	coNP	▽
$\vee$-tgd	Π_3P	▼	DP_2	▼	coNP	
tgd	Π_3P	▲	DP_2	▲	coNP	△
lav tgd	DP	▼	DP	▲	in P	▽
ID	in P	▼	in L	▼	in P	
UC	Π_2P	▼	DP	▼	coNP	
full $\vee$-tgd	Π_2P	▲	DP		coNP	△
full tgd	DP	▼	DP	▲	P	△▽
denial	DP	▼	DP		in L	▽
egd	DP		DP	▲	in L	
FD	in L	▼	in L		in L	
key	in L		in L		in L	

▶ **Theorem 4.** *For all variants of the RC problem studied here, the complexity is as depicted in Table 3. The complexity does not change in the case of bounded arity.*

Known results. In Section 4, we will give the intuition of our results for RC (black triangles in Table 3). In the remainder of this section, we summarize results given in or implicit in previous work (white triangles).

First, we consider RC(D,I) (the second column of Table 3). The PSPACE-hardness for FO follows immediately from the PSPACE-hardness of first-order model checking. We now proceed to RC(C) (i.e., data complexity, the last column of Table 3). Staworko [21] showed coNP-completeness for UCs [21, Corollary 3] and P-membership for full tgds [21, Theorem 2]. A slight modification of the proofs shows that coNP-hardness already holds for full $\vee$-tgds. The matching P-hardness for full tgds was given in [2, Theorem 5], where also coNP-hardness for tgds [2, Theorem 7] and L-membership for denial constraints [2, Proposition 5] was proved. The P-membership for lav tgds was given in [22, Theorem 4.9], and the coNP-membership for FO constraints in [2, Proposition 4].

As a concluding remark, note that in Table 3, we do not separately list RC with bounded arity. A quick inspection of our hardness proofs shows that, in case of the Repair Checking problem, the complexity does not change if the arity is bounded.

4 Repair Checking – Intuition

In this section, we will give the intuition and present selected proofs for the repair checking problem. We first illustrate the sources of complexity by discussing the membership results. After that, we will present selected hardness proofs.

The naive algorithm. Repair checking has two fundamental sources of complexity, namely, checking that the supposed repair I is consistent, and checking that it is indeed minimal. This immediately gives the following naive algorithm:

1. check consistency of I
2. check minimality of I by considering the co-problem, where we try to guess a counter-example to minimality (i.e., a consistent instance with smaller symmetric difference)

Since every database instance with a smaller symmetric difference has size at most polynomial in the size of the input, the guess is polynomial. Thus, if we know that the complexity of model checking of a constraint language is in $\mathcal{C}$, then our naive algorithm yields an upper bound of $\mathcal{C} \wedge \mathsf{coNP}^{\mathcal{C}}$ for the variant of RC where all parts of the input vary. Recall that we write $\mathcal{C}_1 \wedge \mathcal{C}_2$ to denote the conjunctive Boolean combination of two complexity classes.

For $\mathcal{C} = \mathsf{NP}$ (which is the case for lav tgds), it it easy to see that the entire second step fits into coNP. Indeed, for the co-problem, one can simultaneously guess a counter-example (a database instance) and a witness for its consistency. In this case, RC is in DP. For other classes $\mathcal{C}$, the $\mathsf{coNP}^{\mathcal{C}}$ factor dominates.

In many cases, one cannot do better than that. In particular, considering the RC problem (i.e., all parts of the input are allowed to vary), we use the upper bound given by the naive algorithm to show PSPACE-membership for FO, Π_3P-membership for $\vee$-tgds, DP-membership for lav tgds, and Π_2P-membership for UCs (four of the ▼ in the first column of Table 3).

Beyond the naive algorithm. In some cases, one *can* do better than the naive algorithm. For full tgds, [21, Lemma 1] provides an NP test for checking minimality of a candidate repair. Since model checking for full tgds is in coNP, we get a DP algorithm for RC.

For denial constraints, the minimality check only needs to test if all immediate supersets are inconsistent. This is the case since denial constraints can only be repaired by deletions and since they are monotone in the sense that supersets of inconsistent instances are always inconsistent. Since consistency can be checked in coNP for denial constraints, we thus have a DP algorithm for RC. For FDs, the tractability of consistency checking yields a polynomial-time (actually, even a logarithmic-space) algorithm for RC.

For IDs, P-membership for RC exploits the existence of a unique subset repair (subset repairs are those repairs that can be obtained by deletions only). Such a subset repair can be computed in polynomial time in case of IDs. Using a construction similar to the one given in [22, Lemma 4.8], we can exploit subset repairs to devise a polynomial-time algorithm for the RC problem of IDs.

Fixing the instances. If we fix the instances D and I, that is, if we consider $RC(D, I)$, the naive algorithm can be refined into:

1. check consistency
2. for every instance I' between D and I, check $I' \not\models C$

Observe that the second step of the algorithm has turned from a guess into an explicit computation. In total, this refined version yields an upper bound of $\mathcal{C} \wedge \mathsf{co}\mathcal{C}$. Let us now consider the results for RC(D,I) (the second column of Table 3): Using this algorithm, we obtain DP_2-membership for $\vee$-tgds and DP-membership for UCs. For IDs, we can further improve the P upper bound to an L upper bound. This completes our discussion of the membership results shown in Table 3.

Sources of complexity. An inspection of our proofs yields an interesting relationship between the roles of consistency and minimality checking, our two orthogonal sources of complexity. For tgds and full $\vee$-tgds, minimality checking dominates the complexity of RC. In particular, hardness holds even if the given instance is promised to be consistent. In contrast, for our DP results for RC, the role of consistency and minimality checking is distributed between the NP and the coNP parts of DP (i.e. if one check requires NP power and the other one requires coNP power). As a consequence, if the given instance is promised

to be consistent, the complexity of RC for lav tgds drops to coNP while for full tgds and egds it drops to NP.

Selected hardness proofs. We now present two hardness proofs illustrating typical techniques used to obtain our results. Many of our reductions from QSAT problems share similar machinery. We thus define a set and a formula transformation that we will need in most of these proofs. First, we define a set $\hat{c}$ that encodes the legal value-combinations of literals in a clause of a 3CNF formula (i.e., all combinations except for $c(0,0,0)$). Here we identify the truth value true (resp. false) with 1 (resp. 0):

$$\hat{c} = \{c(0,0,1), c(0,1,0), c(0,1,1), c(1,0,0), c(1,0,1), c(1,1,0), c(1,1,1)\}$$

If ψ is a 3CNF formula of the form $\psi = \bigwedge_i (l_{i1} \vee l_{i2} \vee l_{i3})$, then we denote by ψ^* the conjunction $\bigwedge_i c(l^*_{i1}, l^*_{i2}, l^*_{i3})$ where $l^*_{ij} = x$ if l_{ij} is the positive literal x and $l^*_{ij} = \overline{x}$ if l_{ij} is the negative literal $\neg x$. For example, $[(x \vee \neg z \vee y) \wedge (\neg z \vee y \vee \neg y)]^* = c(x, \overline{z}, y) \wedge c(\overline{z}, y, \overline{y})$.

▶ **Lemma 5.** *There is a database instance D such that RC(D) for tgds is Π_3P-hard. This holds even for bounded arity and if it is known that $I \models C$.*

Proof. We proceed by reduction from $\exists$QSAT$_3$ to the co-problem of RC. Let

$$\varphi = \exists x_1 \dots x_k \, \forall y_1 \dots y_l \, \exists z_1 \dots z_m \, \psi$$

be an arbitrary instance of $\exists$QSAT$_3$. W.l.o.g., we may assume that ψ is in 3CNF. From this, we construct an instance (D, I, C) of RC, such that φ is true if and only if I is not a repair of D w.r.t. C.

$$D = \hat{c} \cup \{q(0,1), q(1,0)\} \tag{1}$$

$$I = D \cup \{c(0,0,0)\} \cup \bigcup_{1 \le i \le k} \{p_i(0,1), p_i(1,0)\} \tag{2}$$

$$C = \bigcup_{1 \le i \le k} \{q(x,x') \to \exists y y' \, p_i(y,y')\} \tag{3}$$

$$\cup \bigcup_{1 \le i \le k} \{p_i(x,y) \wedge p_i(y,x) \to c(x,x,x)\} \tag{4}$$

$$\cup \bigcup_{1 \le i \le k} \{q(x,y) \wedge c(x,x,x) \wedge c(y,y,y) \to p_i(x,y)\} \tag{5}$$

$$\cup \{ \bigwedge_{1 \le i \le k} p_i(x_i, \overline{x}_i) \wedge \bigwedge_{1 \le i \le l} q(y_i, \overline{y}_i) \to \exists z_1 \overline{z}_1 \dots z_m \overline{z}_m \bigwedge_{1 \le i \le m} q(z_i, \overline{z}_i) \wedge \psi^* \} \tag{6}$$

It is easy to see that I is a superset of D that is consistent with C. So we claim that φ is true if and only if there is a consistent instance I' with $D \subseteq I' \subsetneq I$. The constraints restrict such an instance I' to a specific form:

1. I' does not contain $c(0,0,0)$: otherwise by (5) it would contain all of I.
2. I' contains exactly one of $p_i(1,0)$ and $p_i(0,1)$ for all $i \le k$: by (3) I' contains at least one, and (4) would add $c(0,0,0)$ if more than one were present.

The second property establishes a natural 1-to-1 correspondence between such instances and truth assignments to the x_i variables: instance I' corresponds to truth assignment μ with

$$\mu(x_i) = \begin{cases} T & \text{if } p_i(1,0) \in I' \\ F & \text{if } p_i(0,1) \in I' \end{cases}$$

and vice versa. Finally note that I' satisfies C, and in particular the last constraint (6), if and only if $\forall \vec{y} \exists \vec{z} \, \psi$ is satisfied by the assignment μ corresponding to I'. ◀

We note that while these constraints are not weakly acyclic (see e.g. [22]), the proof can be easily adapted to turn the constraints into a weakly acyclic set of tgds.

▶ **Lemma 6.** *There are database instances D, I such that RC(D,I) for egds is* DP*-hard. This holds even for bounded arity. If we know that $I \models C$, then this drops to* NP*-hard.*

Proof. We proceed by reduction from 3-colorability and its complement. Let $G = (V, E)$ be an arbitrary instance of 3-colorability and $G' = (V', E')$ be an arbitrary instance of not-3-colorability. W.l.o.g., assume that both E and E' contain the edge $(1, 2)$. From this, we construct the following instance (D, I, C) of RC.

$$D = \{b(1,2), b(1,3), b(2,3), b(2,1), b(3,1), b(3,2), g\}$$
$$I = D \setminus \{g\}$$
$$C = \{ \bigwedge_{(i,j)\in E} b(x_i, x_j) \wedge g \to x_1 = x_2, \quad \bigwedge_{(i,j)\in E'} b(x_i, x_j) \to x_1 = x_2\}$$

Observe how the big conjunctions encode the graphs, and can be satisfied if and only if the corresponding graph is 3-colorable. Since both graphs contain an edge between the vertices 1 and 2, the atom $b(x_1, x_2)$ appears in both conjunctions, ensuring that x_1 and x_2 are assigned different values and thus that the right-hand sides of the egds are false.

Therefore, I is consistent iff G' is not 3-colorable, and D is consistent (thus I not minimal) iff neither G nor G' are 3-colorable. In sum, I is a repair of D w.r.t. C iff G is 3 colorable and G' is not. ◀

5 Consistent Query Answering – Intuition

In this section, we will give the intuition and present selected proofs for the consistent query answering problem. As in the previous section, we first illustrate the sources of complexity by discussing the membership results. After that, we will present selected hardness proofs.

Existential constraints. We first consider existential constraints, i.e., all classes of constraints that allow existential quantification in the conclusion (in Figure 1, these can be found on the left-most branch from FO to ID). For these classes of constraints, we see a particularly clear-cut picture of complexity: They are either undecidable, or have relatively low complexity (in P or NP-complete, depending on the type of complexity considered). The reason for this sharp contrast in complexity is the following: by the monotonicity of CQs, the relevant repairs for CQ-answering are the subset-minimal ones. In case of lav tgds and IDs, we can be sure that all subset-minimal repairs are subsets of the original database instance D. This property is lost for tgds as the following simple example shows:

▶ **Example 7.** Consider the instance (D, C, Q) of CQA with $D = \{a, b\}$ and $C = \{a \to e;\ b \wedge e \to f\}$ and $Q = b$. In this case, the minimal repairs are $\{b\}$ and $\{a, e\}$. Then we have $\{a, e\} \not\models Q$ and, therefore, $(D, C) \not\models Q$ even though Q is satisfied in all subset-repairs of D w.r.t. C (actually, $\{b\}$ is the only subset-repair). The difficulty comes from the fact that deleting b in a repair only makes sense after e has been added. Such an effect cannot occur with lav tgds. ◀

Consequently, even though lav tgds and IDs also contain existential quantification, all variants of CQA yield complexity of at most NP-completeness. Looking at Table 1, one can see that while known results showed essentially identical pictures for lav tgds and IDs (e.g., CQA is

NP-complete for both, CQA(C,Q) is in P for both), it turns out that for CQA(Q) as well as CQA(D,Q) we have NP-completeness for lav tgds while for IDs we have P-membership. The intuitive reason for P-membership of CQA(Q) with IDs is the existence of a unique subset repair which can be computed in polynomial time for IDs (but the computation requires NP power for lav tgds).

In contrast, for tgds and all extensions thereof (i.e., $\vee$-tgds and FO constraints) undecidability holds for all types of complexity considered here. That is, undecidability holds even if only *one* of the three parts of the input is allowed to vary. Note that there is a close relationship between the CQA problem and the problem of CQ-answering under tgds. In the latter problem, we are given a database D, a set C of tgds and a conjunctive query Q. The question is if D (considered as a conjunction of ground atoms) together with C logically implies Q. It is well-known that the latter problem is undecidable even if (D,Q) or (C,Q) is fixed [15, 7]. From these undecidability results, the undecidability of CQA(D,Q) and of CQA(C,Q) follows immediately. To the best of our knowledge, the undecidability of CQ-answering for fixed (D,C) has not been published so far. It has been observed by G. Gottlob [12] independently of our undecidability proof for CQA(D,C). The key idea of the latter proof is that even for fixed D and C, there can exist arbitrarily big repairs. We can then encode the Halting problem into the CQA(D,C) problem via CQs that ask for the existence of certain chains of binary atoms in every repair.

From universal constraints to full tgds. For universal constraints, the intuition of membership in many cases originates from our algorithm for UCs that we will present next. Previous work [3] showed coN2EXP-membership for CQA (i.e., all parts of the input vary). The algorithm we present here yields a $\mathsf{coNEXP}^{\mathsf{NP}} = \Pi_2\mathsf{EXP}$ upper bound, which together with our hardness results allows us to establish completeness in all cases. Note that [3] considers a semantical definition of UCs that includes logically equivalent FO formulas. Our algorithm also applies to their setting, thus closing the gap left as future work in that paper.

We first illustrate the key ideas of the $\mathsf{coNEXP}^{\mathsf{NP}}$-membership proof for UCs. Let (D, C, Q) be an instance of CQA. The crucial observation is that it never makes sense to introduce fresh domain elements when repairing w.r.t. UCs. More precisely, let G be the set of all ground atoms over the active domain of D. Then every repair of D is a subset of G. We now give the following $\mathsf{NEXP}^{\mathsf{NP}}$ algorithm for the co-problem of CQA.

1. Guess $I \subseteq G$
2. Check that $I \not\models Q$ and $I \models C$
3. Call an oracle to check that there is no J such that $J \models C$ and that J has smaller symmetric difference to D than I

For verifying the complexity of our algorithm, observe that G has at most exponential size. Furthermore, note that checking whether a first-order formula φ is satisfied by a model M can be done in time $O\left(|\varphi|^2 \times |M|^{|\varphi|}\right)$. This model-checking algorithm can also be used inside the NP oracle by padding its input. Thus our algorithm is indeed in $\mathsf{NEXP}^{\mathsf{NP}}$. The cost of the exponential guess in the first step and the call to an NP oracle in the last step remains unchanged even if D and Q are fixed. In contrast, if D and C are fixed, then the complexity drops to NP, i.e., the query complexity of CQ-answering.

Full $\vee$-tgds fall into exactly the same classes of complexity as UCs for all types of complexity – in essence, membership holds for UCs while our hardness results only use full $\vee$-tgds. For full tgds, inspired by [21, Lemma 1], a refined algorithm that exploits the limited number of repairs yields EXP-membership for CQA. Again, the main sources of complexity persist even if D and Q are fixed.

Bounded arity. In Table 2, we observe that only for UCs, full $\vee$-tgds, and full tgds the complexity decreases if we assume bounded arity of all relation symbols involved. More precisely, the complexity drops from the exponential hierarchy (Table 1) to the polynomial hierarchy (Table 2). The reason for this is that if the arity of the relation symbols is bounded by a constant, the number of possible ground atoms over the active domain, and therefore the size of any repair, is polynomially bounded in the size of the input.

The $\Pi_3 P$ upper bound for UCs is obtained by simply revisiting the basic algorithm above and using the fact that now the guess of $I \subseteq G$ in the first step is polynomially bounded. For the repair check (i.e., mainly the minimality check in the third step), a Π_2P oracle is needed. These two sources of complexity persist even for CQA(D,Q).

Also for full tgds, the upper bound is obtained via the basic algorithm. In this case, the repair check drops to DP. We thus get the Π_2P upper bound for CQA with full tgds. In contrast to full $\vee$-tgds, the complexity decreases if we fix the database D. It is convenient to consider a set of full tgds as a datalog program. Then the minimality check only requires the computation of the least fixed point of the immediate consequence operator defined by the datalog program for all (constantly many) subsets of the fixed database D. This fixed point computation can be done with polynomially many nonadaptive oracle calls to an NP oracle. This gives us the Θ_2P upper bound in the last line of Table 2.

Less expressive subclasses of UCs. In case of FDs, we have NP-membership for CQA(D), since we can exploit that all repairs are subsets of (a fixed) D. The same holds for the (more expressive) denial constraints, but here we have BH-membership: since model checking is now coNP-hard, a single NP call is not sufficient – but as the subsets are again fixed, no more than constantly many NP oracle calls are needed. The remaining L-membership for CQA(D,Q) with FDs, which we distinguish because known results [2] feature such a more fine-grained analysis, follows from the fact that all repairs are subsets, and repair checking is possible in L.

Selected hardness proofs. The following hardness proof – establishing hardness for any level in the Boolean hierarchy – is of a significantly different flavor than the proofs in Section 4. We will then also present a $\Pi_3 P$-hardness proof, which uses similar ideas as the proofs in Section 4.

The levels of the Boolean hierarchy BH are denoted by BH_k. In the proof that follows, we will be interested in the co-classes coBH_k that can be defined as follows: $\mathsf{coBH}_1 = \mathsf{coNP}$, $\mathsf{coBH}_2 = \mathsf{coBH}_1 \vee \mathsf{NP}$, $\mathsf{coBH}_3 = \mathsf{coBH}_2 \wedge \mathsf{coNP}$, $\mathsf{coBH}_4 = \mathsf{coBH}_3 \vee \mathsf{NP}$, and so on.

▶ **Lemma 8.** *There is an atomic query Q, such that for every $k > 0$ there is a database instance D s.t. CQA(D,Q) for egds is BH_k-hard. This holds even for bounded arity.*

Proof. We reduce from a coBH_k-hard problem. This suffices since $\mathsf{BH}_k \subseteq \mathsf{coBH}_{k+1}$. As our coBH_k-hard problem, we consider the Boolean combination of 3-colorability. Thus an instance of our problem is given by k graphs $G_1, G_2, \ldots, G_k$ with edges E_i. The question of our problem is whether the Boolean combination of 3-colorability is true. W.l.o.g. let all graphs $G_1, G_2, \ldots, G_k$ contain the edge $(1, 2)$.

We now construct our equivalent CQA(D,Q) instance (D_k, C_k, Q). We first construct Q and D_k. Note that Q is fixed and D_k depends only on k and not on the graphs given in the

input of our problem.

$$Q = \exists z_1 z_2 z_3 \; a(z_1, z_2, z_3)$$
$$D_k = \{a(1,2,3)\} \cup \bigcup_{1 \leq n \leq k} \{b_n(1,2), b_n(1,3), b_n(2,3), b_n(2,1), b_n(3,1), b_n(3,2)\}$$

Intuitively, the query asks whether there is an a-tuple, and the database D_k consists of an a-tuple and all valid color combinations.

We now proceed to constructing C_k, which will encode the Boolean combination of the k instances of 3-colorability. Recall that the definition of coBH_k alternates between odd ($\mathsf{coBH}_k = \mathsf{coBH}_{k-1} \wedge \mathsf{coNP}$) and even ($\mathsf{coBH}_k = \mathsf{coBH}_{k-1} \vee \mathsf{NP}$) cases. This alternation will be reflected in the construction of C_k. It is convenient to define the following abbreviation: $Col_n = b_n(y_1, y_2) \wedge b_n(y_1, y_3) \wedge b_n(y_2, y_3) \wedge b_n(y_2, y_1) \wedge b_n(y_3, y_1) \wedge b_n(y_3, y_2)$.

$$O_0 = A_0 = \emptyset$$
$$A_n = \begin{cases} A_{n-1} \cup \{a(y_1, y_2, y_3) \wedge \bigwedge_{(i,j) \in E_n} b_n(x_i, x_j)\} & n \text{ odd} \quad (1) \\ \{\varphi \wedge Col_n \mid \varphi \in A_{n-1}\} & n \text{ even} \quad (2) \end{cases}$$
$$O_n = \begin{cases} O_{n-1} & n \text{ odd} \\ O_{n-1} \cup \{\bigwedge_{(i,j) \in E_n} b_n(x_i, x_j)\} & n \text{ even} \quad (3) \end{cases}$$
$$C_k = \{\varphi \to x_1 = x_2 \mid \varphi \in A_k \cup O_k\}$$

For illustration, here is C_2:

$$\{a(y_1, y_2, y_3) \wedge \bigwedge_{(i,j) \in E_1} b_1(x_i, x_j) \wedge Col_2 \to x_1 = x_2, \; \bigwedge_{(i,j) \in E_2} b_2(x_i, x_j) \to x_1 = x_2\}$$

Recall from Lemma 6 how $x_1 = x_2$ in the conclusions can never be fulfilled and thus effectively represent negative constraints. Intuitively, the formula constructed in A_n in the odd case (1) provides a reason to delete $a(1,2,3)$ iff the graph G_n is 3-colorable. The formula constructed in O_n in the even case (3) – together with the additional Col_n conjuncts added in (2) – nullifies any reason to delete $a(1,2,3)$ iff the graph G_n is 3-colorable.

We now proceed to the correctness proof. Let us first note that the query is false in a repair R if and only if there is a formula φ in A_k that is satisfied by $R \cup \{a(1,2,3)\}$. The proof goes by induction and in two cases.

1. k is odd ($\mathsf{coBH}_k = \mathsf{coBH}_{k-1} \wedge \mathsf{coNP}$)
   In this case, there is only one formula added, namely in (1). If G_k is 3-colorable, then this formula is satisfied by D_k and one can avoid deleting one of b_k by deleting $a(1,2,3)$. If on the other hand G_k is not 3-colorable, then the formula cannot be satisfied. So there is a repair of D_k w.r.t. C_k that falsifies the query if and only if there is such a repair of D_{k-1} w.r.t. C_{k-1}. For the base case $k = 1$ note that D_0 is already consistent.
2. k is even ($\mathsf{coBH}_k = \mathsf{coBH}_{k-1} \vee \mathsf{NP}$)
   In this case, there is only one formula added, namely in (3). If G_k is 3-colorable then one of b_k has to be deleted as enforced by the formula. Yet then, by the modifications in (2), no constraint from A_k can ever fire and delete a. On the other hand if G_k is not 3-colorable, then the added formula cannot be satisfied. So there is a repair of D_k w.r.t. C_k that falsifies the query if and only if there is such a repair of D_{k-1} w.r.t. C_{k-1}. ◀

▶ **Lemma 9.** *There is a database instance D and an atomic query Q, s.t. CQA(D,Q) for full $\vee$-tgds is $\Pi_3\mathsf{P}$-hard in case of bounded arity.*

Proof. We proceed by reduction from $\exists$QSAT$_3$ to the co-problem of CQA(D,Q). Let

$$\varphi = \exists x_1 \dots x_k \, \forall y_1 \dots y_l \, \exists z_1 \dots z_m \, \psi$$

be an arbitrary instance of $\exists$QSAT$_3$. W.l.o.g., we may assume that ψ is in 3CNF. From this, we construct the following instance (D, C, Q) of CQA, where $\hat{c}$ and ψ^* are as defined in the paragraph preceding Lemma 5.

$$D = \hat{c} \cup \{r(0,1), r(1,0), d(0,1), a\} \tag{1}$$

$$C = \bigcup_{1 \le i \le k} \{d(x,y) \to p_i(x,y) \vee p_i(y,x)\} \tag{2}$$

$$\cup \bigcup_{1 \le i \le l} \{d(x,y) \to q_i(x,y) \vee q_i(y,x)\} \tag{3}$$

$$\cup \bigcup_{1 \le i \le l} \{d(x,y) \wedge b \to q_i(x,y) \wedge q_i(y,x)\} \tag{4}$$

$$\cup \{ \bigwedge_{1 \le i \le k} p_i(x_i, \overline{x_i}) \wedge \bigwedge_{1 \le i \le l} q_i(y_i, \overline{y_i}) \wedge \bigwedge_{1 \le i \le m} r(z_i, \overline{z_i}) \wedge \psi^* \to b\} \tag{5}$$

$$\cup \{d(x,y) \wedge a \wedge b \to e\} \tag{6}$$

$$Q = a \tag{7}$$

We claim that φ is true iff there is a repair R of D w.r.t. C in which Q is false. Clearly, D is inconsistent since it violates the first $\vee$-tgd in line (2). We distinguish two main cases of repairs, namely either $d(0,1)$ is deleted from D or $d(0,1)$ is retained. If $d(0,1)$ is deleted then there is no reason to delete a. Hence, in these repairs, Q is clearly true. Hence, the only interesting case are repairs which do contain $d(0,1)$.

A repair R containing $d(0,1)$ also contains exactly one of $\{p_i(0,1), p_i(1,0)\}$ for every $i \in \{1, \dots, k\}$. We thus get a 1-to-1 correspondence between the choice of p_i-atoms and truth assignments on $\{x_1, \dots, x_k\}$. Similarly, by the $\vee$-tgd in line (3), at least one of $\{q_i(0,1), q_i(1,0)\}$ for every $i \in \{1, \dots, l\}$ has to be added to R. In case exactly one of $\{q_i(0,1), q_i(1,0)\}$ is added to R, we again get a 1-to-1 correspondence between the choice of q_i-atoms and truth assignments on $\{y_1, \dots, y_l\}$.

Now the crucial question is whether the tgd in line (5) fires and b has to be added to R. If so, then for every $i \in \{1, \dots, l\}$ both $q_i(0,1)$ and $q_i(1,0)$ have to be added to R, due to the tgd in line (4). Note that R thus contains a strict superset of all other choices of q_i-atoms. Due to the minimality of R, the tgd in line (5) thus encodes the following condition: for the chosen p_i-atoms, no matter how we choose one q_i-atom for every $i \in \{1, \dots, l\}$, there exists an instantiation of the variables $(z_i, \overline{z_i})$ to $(0,1)$ or $(1,0)$, such that ψ^* can be matched into the repair R. Finally, note that, since a only occurs in line (6), a repair that deletes a has to contain b.

By making use of the correspondence between p_i and q_i atoms in R on the one hand, and truth assignments to the variables in φ on the other hand, we get the desired equivalence, namely: $(D, C) \not\models Q$ iff there exists a repair R with $a \notin R$ iff there exists a truth assignment μ on $\{x_1, \dots, x_k\}$, s.t. for every extension of μ to $\{y_1, \dots, y_l\}$, there exists a further extension ν to the variables $\{z_1, \dots, z_m\}$, s.t. $\nu \models \psi$, i.e., φ is true. ◀

6 Conclusion

In this work, we have provided a complete picture of the complexity of the RC- and CQA-problems for a wide range of constraint languages. While previous work provided important

parts of the picture (in particular, a thorough analysis of data complexity), this work now completes the picture for all types of complexity. In many cases, this has allowed us to get a better understanding of the true sources of complexity.

Tables 1 and 2 summarize the picture for consistent query answering, while Table 3 does the same for repair checking. In particular, for the CQA problem, we get a great variety of complexity results ranging from tractability via various levels of the polynomial hierarchy and various results in the exponential hierarchy up to undecidability. We observe several similarities between classes of constraints such as for UCs and $\vee$-tgds (also for denial constraints and egds). On the other hand, in several cases, we see a diversity of complexity in settings where results for data complexity where relatively uniform. For example, for full tgds, as well as denial constraints and their three subclasses, previously known data complexity results (i.e., CQA(C,Q)) showed coNP-completeness in all five cases (cf. the fifth column of Table 1). Yet if we consider different types of complexity, we see a much more diverse picture, e.g. if we look at CQA(Q) (i.e., the query Q is fixed): full tgds are EXP-complete (Π_2P-complete for bounded arity), while denial constraints and egds are Π_2P-complete and FDs and key dependencies remain coNP-complete (cf. the fourth column of Table 1).

Similar stories could be told about other types of complexity in our tables (for example, in the second column of Table 1 we get BH or Θ_2P instead of Π_2P and we get NP instead of coNP) or for different types of constraint classes considered here. In total, we believe that apart from completing the picture for all types of complexity, the results obtained give new insights into the sources of complexity that were hidden before.

Future work. In this work, we have considered a wide range of constraint languages. However, in the literature, further classes of constraint languages can be found, such as binary constraints [9], weakly acyclic tgds [22], and further subclasses of tgds [17]. The exploration of the combined complexity of the RC- and CQA-problems for ICs from these classes has been left for future work.

Yet more importantly, settings with combinations of various kinds of constraints (such as, e.g., inclusion dependencies with key dependencies) should be further explored, thus extending work that was already started in [8]. Finally, further problem variants deserve future investigation, such as adopting different notions of repairs either by restricting the allowed repair actions or by considering different notions of minimality.

Another natural next question is what happens if not only the arity is bounded, but the whole schema is fixed. In most cases this does not seem to change anything, but some problems (especially regarding UCs and full $\vee$-tgds) indeed become easier. Consider for example RC(D) for UCs: Here, since the schema and the active domain are fixed, all repairs are among a constant set of instances. This allows for an NP minimality check, so the complexity of RC(D) drops to DP - in stark contrast to all cases we considered, where RC(D) and RC always have the same complexity.

Acknowledgments. This work was supported by the Austrian Science Fund (FWF):P25207-N23 and Y698 and by the Vienna Science and Technology Fund (WWTF) project ICT12-15. Sebastian Arming is currently supported by the Austrian Science Fund (FWF): S11411-N23. Emanuel Sallinger is currently supported by the EPSRC grant EP/M025268/1.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://www-cse.ucsd.edu/users/vianu/book.html`.

2 Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41. ACM Press, 2009. doi:10.1145/1514894.1514899.

3 Marcelo Arenas and Leopoldo Bertossi. On the decidability of consistent query answering. In *AMW*, 2010. URL: http://ceur-ws.org/Vol-619/paper10.pdf.

4 Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999.

5 Leopoldo Bertossi. Consistent query answering in databases. *ACM SIGMOD Record*, 35(2):68, 2006. doi:10.1145/1147376.1147391.

6 Leopoldo Bertossi. Database Repairing and Consistent Query Answering. *Synthesis Lectures on Data Management*, 2011. doi:10.2200/S00379ED1V01Y201108DTM020.

7 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research (JAIR)*, 48:115–174, 2013.

8 Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271. ACM Press, 2003. doi:10.1145/773153.773179.

9 Jan Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, pages 1–17. Springer, 2007. doi:10.1007/11965893_1.

10 Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005. doi:10.1016/j.ic.2004.04.007.

11 Gaëlle Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? *ACM Trans. Comput. Log.*, 16(1):7:1–7:24, 2015. doi:10.1145/2699912.

12 Georg Gottlob. Personal communication, 2015.

13 Sergio Greco, Fabian Pijcke, and Jef Wijsen. Certain query answering in partially consistent databases. *PVLDB*, 7(5):353–364, 2014. URL: http://www.vldb.org/pvldb/vol7/p353-greco.pdf.

14 David S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 67–161. MIT Press, 1990.

15 David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences (JCSS)*, 28(1):167–189, 1984.

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, 2015. doi:10.1145/2745754.2745769.

17 Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. From classical to consistent query answering under existential rules. In *AAAI*, pages 1546–1552. AAAI Press, 2015.

18 Carsten Lutz and Frank Wolter. On the relationship between consistent query answering and constraint satisfaction problems. In *ICDT*, pages 363–379, 2015. doi:10.4230/LIPIcs.ICDT.2015.363.

19 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

20 Andreas Pfandler and Emanuel Sallinger. Distance-bounded consistent query answering. In *IJCAI*, pages 2262–2269, 2015. URL: http://ijcai.org/papers15/Abstracts/IJCAI15-320.html.

21 Sławomir Staworko and Jan Chomicki. Consistent query answers in the presence of universal constraints. *Information Systems*, 35(1):1–22, 2010. doi:10.1016/j.is.2009.03.004.

22 Balder ten Cate, Gaëlle Fontaine, and Phokion G. Kolaitis. On the data complexity of consistent query answering. *Theory Comput. Syst.*, 57(4):843–891, 2015. doi:10.1007/s00224-014-9586-0.

23 Jef Wijsen. A survey of the data complexity of consistent query answering under key constraints. In *FoIKS*, pages 62–78, 2014. doi:10.1007/978-3-319-04939-7_2.

On the Complexity of Enumerating the Answers to Well-designed Pattern Trees

Markus Kröll[1], Reinhard Pichler[2], and Sebastian Skritek[3]

1 **TU Wien, Vienna, Austria**
`kroell@dbai.tuwien.ac.at`
2 **TU Wien, Vienna, Austria**
`pichler@dbai.tuwien.ac.at`
3 **TU Wien, Vienna, Austria**
`skritek@dbai.tuwien.ac.at`

Abstract

Well-designed pattern trees (wdPTs) have been introduced as an extension of conjunctive queries to allow for partial matching – analogously to the OPTIONAL operator of the semantic web query language SPARQL. Several computational problems of wdPTs have been studied in recent years, such as the evaluation problem in various settings, the counting problem, as well as static analysis tasks including the containment and equivalence problems. Also restrictions needed to achieve tractability of these tasks have been proposed. In contrast, the problem of enumerating the answers to a wdPT has been largely ignored so far. In this work, we embark on a systematic study of the complexity of the enumeration problem of wdPTs. As our main result, we identify several tractable and intractable cases of this problem both from a classical complexity point of view and from a parameterized complexity point of view.

1998 ACM Subject Classification H.2.3 Database Management – Languages

Keywords and phrases SPARQL, Pattern Trees, CQs, Enumeration, Complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.22

1 Introduction

With the steadily increasing amount of inaccurate and incomplete data on the web, the need for partial matching as an extension of Conjunctive Queries (CQs) is gaining more and more importance. Therefore, in the semantic web query language SPARQL, the OPTIONAL operator is a crucial feature. Recently, this operator (which we shall abbreviate to "OPT" in the sequel) has also been studied for arbitrary relational vocabulary [4]. Intuitively, the OPT operator (which corresponds to the left outer join in the Relational Algebra) allows the user to extend CQs by optional parts which are retrieved from the data if available, but which do not cause the partial answers to get lost in case the optional information is not available. The following example will help to illustrate this idea.

▶ **Example 1.** Consider the following {AND,OPT}-SPARQL query that is posed over a database that stores information about movies:[1]

$$\Big(\big((x, \texttt{directed\_by}, y)\ \mathsf{AND}\ (x, \texttt{released}, \text{“before\_1980”})\big)\ \mathsf{OPT}\ (x, \texttt{oscars\_won}, z)\Big)\ \mathsf{OPT}\ (y, \texttt{first\_movie}, z').$$

[1] We use here the algebraic-style notation from [25] rather than the official SPARQL syntax of [28].

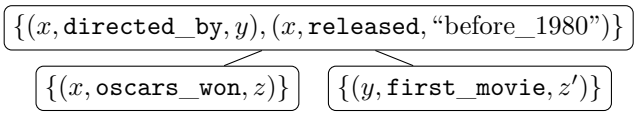


Figure 1 The wdPT representing the query from Example 1.

This query retrieves all pairs (m, d) s.t. movie m is directed by d and released before 1980. This is specified by the pattern $(x, \texttt{directed\_by}, y)$ AND $(x, \texttt{released}, \text{“before\_1980”})$. Furthermore, whenever possible, this query also retrieves (one or both of) the following pieces of data: the number of Academy Awards n won by movie m and the first movie m' directed by d. In other words, in addition to (m, d) we also retrieve n and/or m' if the information is available in the database. This is specified by the atoms $(x, \texttt{oscars\_won}, z)$ and $(y, \texttt{first\_movie}, z')$ following the respective OPT-operators.

In [23], so-called *well-designed pattern trees* (wdPTs) were introduced as a convenient graphical representation of CQs extended by the OPT operator. Intuitively, the nodes in a wdPT correspond to CQs while the tree structure represents the optional extensions. The wdPT corresponding to the query in Example 1 is displayed in Figure 1.

The semantics of a wdPT p is defined in terms of the CQs contained in it: with each subtree T' of T containing the root, we associate a CQ $q_{T'}$ defined by the conjunction of all atoms in the nodes of T'. The evaluation of wdPT p over database $\mathcal{D}$ consists then of all “maximal” answers to the CQs of the form $q_{T'}$. That is, we take the union of all answers to the CQs of the form $q_{T'}$, for each T', and then remove all those answers that are “extended” by some other answer in the set. We revisit Example 1 to illustrate these ideas.

▶ **Example 2.** Consider an RDF database $\mathcal{D}$ consisting of triples (“American_Graffiti”, `directed_by`, “George_Lucas”), (“American_Graffiti”, `released`, “before_1980”), (“Star_Wars”, `directed_by`, “George_Lucas”), (“Star_Wars”, `released`, “before_1980”), (“Star_Wars”, `oscars_won`, “6”). The evaluation over $\mathcal{D}$ of the wdPT in Figure 1, and, therefore, of the query from Example 1, consists of partial mappings μ_1 and μ_2 defined on variables x, y, z, z' such that: (1) μ_1 is only defined on x and y in such a way that $\mu_1(x) =$ “American_Graffiti” and $\mu_1(y) =$ “George_Lucas”, and (2) μ_2 is defined on x, y and z in such a way that $\mu_2(x) =$ “Star_Wars”, $\mu_2(y) =$ “George_Lucas”, and $\mu_2(z) =$ “6”.

To make wdPTs a proper extension of CQs, wdPTs have to be enhanced with projection. For instance, for the wdPT in Example 1, one might decide to project out the variable x. In this way, the answers μ_1 and μ_2 over the database in Example 2 would be restricted to μ_1' and μ_2', s.t. (1) μ_1' is only defined on y with $\mu_1'(y) =$ “George_Lucas”, and (2) μ_2' is defined on y and z and it holds that $\mu_2'(y) =$ “George_Lucas” and $\mu_2'(z) =$ “6”.

Several aspects of the complexity of wdPTs have been studied in previous works. Given a database $\mathcal{D}$ and a query p, the evaluation problem asks if some given mapping μ is an answer to p over $\mathcal{D}$. This problem was shown coNP-complete for wdPTs without projection [25] and Σ_2^P-complete for wdPTs with projection [23]. Wrapping wdPTs into the CONSTRUCT operator of SPARQL makes the evaluation problem NP-complete [20]. In [2], the max-evaluation problem was introduced as a variant of the evaluation problem. Note that if we allow projection, then it may happen that both, some mapping μ and a proper extension of μ are solutions. This is indeed the case for μ_1' and μ_2' above. In [2], the *max-evaluation* problem was identified as an important variant of wdPT evaluation: it asks if a given mapping is a *maximal* solution (i.e., it cannot be properly extended to another solution). This problem is DP-complete [2]. In [27], the counting problem of wdPTs was studied, i.e., given a database

$\mathcal{D}$ and a query p, what is the number of solutions? It turned out that the counting problem of wdPTs remains intractable even under very severe restrictions. Important query analysis tasks such as the containment and equivalence problems in various settings were studied in [26]. It was shown that the complexity of these tasks ranges from NP-completeness (if we disallow projection) to undecidability (if projection is allowed in both queries).

Another interesting and very natural problem in the context of query languages is the *enumeration problem* (i.e., computing one solution after the other without ever outputting duplicates). Despite recent interest in the enumeration problem for First-Order and Conjunctive Queries [5, 19, 29, 12], the complexity of enumerating the answers to wdPTs has been hardly considered so far. A notable exception is [23], where it was shown that enumeration of wdPTs without projection is tractable provided that the CQs contained in the wdPTs are from some tractable fragment. In contrast, for wdPTs with projection, the enumeration problem of wdPTs was shown intractable in [23] even when allowing only acyclic CQs to appear at each node of a wdPT.

Goal. The goal of this work is to initiate a systematic complexity study of the enumeration problem of wdPTs. We thus view the enumeration problem from various angles:

- *Classical complexity analysis of the combined complexity.* We aim at identifying the boundary between tractable and intractable enumeration for the set of both, *all* solutions and *maximal* solutions. In [17], various notions of tractable enumeration are defined. We concentrate on two such notions, namely polynomial delay as the strongest and output polynomial time as the weakest form of tractability. The former means that the time before outputting the first solution, the time between any two solutions and the time between the last output and termination are all bounded by a polynomial in the size of the input. The latter means that the total time for outputting all solutions is bounded by a polynomial in the combined size of the input *plus* the output. In our quest for tractable enumeration, we start with restrictions introduced in [4] to achieve tractable evaluation and we will introduce further restrictions to get a better understanding of the sources of complexity.
- *Parameterized complexity.* Over more than a decade, parameterized complexity theory [11] has developed into a well-established approach to dealing with intractability. The ideal result of a parameterized complexity analysis is fixed-parameter tractability. This means that the problem at hand can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the size of the input and $f(k)$ is a function depending solely on the parameter k. In other words, the exponential explosion can thus be confined to the parameter. By FPT, we denote the class of problems with this behavior. The opposite behavior is fixed-parameter intractability. This means that we can at best achieve an upper bound $\mathcal{O}(n^{f(k)})$ on the time complexity. In other words, the size of the parameter occurs in the exponent of the input size n. In this paper, we take the size of the query as parameter. In a sense, this gives us an intermediate state between data complexity (where the query is considered as fixed) and classical combined complexity (where the query is treated equally as the data). The parameterized complexity approach allows us to obtain a more fine-grained picture of the intractable cases by exploring the boundary between FPT-delay and fixed-parameter intractable enumeration.
- *Revisiting the evaluation problem.* Our complexity analysis of the enumeration problem will bring to light an interesting relationship between fixed-parameter tractable *enumeration* and *evaluation*. More formally, we will show that FPT-delay of the enumeration problem for some type of wdPTs implies that also the evaluation problem for this type

of wdPTs is in FPT. We thus resume the quest for tractable evaluation from [4] and inspect the intractable cases (classical complexity) from a parameterized complexity point of view. We thus aim at delineating the border between fixed-parameter tractability and fixed-parameter intractability to get a better understanding of the nature of the intractability.

- *Data complexity.* Finally, we have a brief look at the data complexity of the enumeration problem of wdPTs. Note that polynomial delay is trivial for query languages derived from First-Order logic. Hence, the goal of analyzing the data complexity is to construct an enumeration algorithm which works with linear-time preprocessing (i.e., the time to produce the first output) and constant delay (i.e., the time between any two successive outputs and the time after the last output).

Organization of the paper and summary of results. In Section 2, we recall some basic notions and results. A conclusion and an outlook to future work are given in Section 6. The main results of the paper are detailed in Sections 3–5, namely:

- *Evaluation problem.* In Section 3, we revisit the evaluation problem of wdPTs. More precisely, we subject the intractable cases from [4] for the evaluation problem to a parameterized complexity analysis. By establishing both fixed-parameter tractability and intractability results, we provide a more fine-grained picture of the complexity of this problem.
- *Combined complexity.* In Section 4, we study the combined complexity of enumerating *all* solutions and of enumerating the *maximal* solutions. We have already recalled above that – without any restrictions – testing if some mapping is a solution is harder than testing if it is a *maximal* solution [23, 2]. By the same token, it was shown in [4] that strictly more severe restrictions on the wdPTs are needed to achieve tractability of the evaluation problem than for the max-evaluation problem. Our complexity analysis of the enumeration problem reveals an interesting effect: it turns out that enumerating the maximal solutions is harder than enumerating all solutions. A yet more detailed picture of the complexity of the enumeration problem (for all solutions resp. for the maximal solutions) is provided by studying also the parameterized complexity of this problem under various restrictions.
- *Data complexity.* In Section 5, we study the data complexity of the enumeration problem (for all resp. for the maximal solutions) of wdPTs. Under the common assumption that matrix multiplication of two Boolean $n \times n$ matrices is not feasible in time $\mathcal{O}(n^2)$, we rule out the possibility of *linear* time preprocessing and constant delay for a very restricted class of wdPTs. However, if we allow *polynomial* time preprocessing then constant delay is achievable for an appropriately restricted class of wdPTs.

2 Preliminaries

Conjunctive queries. We assume familiarity with the relational model, especially with *conjunctive queries (CQs)*, see [1] for details. In the following, we fix some notation. For a set $\mathcal{A}$ of atoms we use $dom(\mathcal{A})$ to denote the set of constants and variables appearing in $\mathcal{A}$, while $var(\mathcal{A})$ refers to the variables only. Similarly, for a mapping μ we denote with $dom(\mu)$ the set of elements on which μ is defined. It is convenient to denote mappings as sets of ordered pairs. Thus two mappings μ_1, μ_2 are equal if $\mu_1 = \mu_2$, and a mapping μ_2 extends a mapping μ_1 if $\mu_1 \subseteq \mu_2$. Furthermore, μ_2 is a proper extension of μ_1 if $\mu_1 \subset \mu_2$.

We write CQs q as $\text{Ans}(\vec{x}) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$, where $\vec{x}$ are the *free variables*, and define $var(q) = var(\{R_i(\vec{v}_i) \mid 1 \leq i \leq m\})$. For the *evaluation* $q(\mathcal{D})$ of a CQ q with free variables $\vec{x}$ over database $\mathcal{D}$, we depart slightly from the traditional definition. We define $q(\mathcal{D})$ to be the set of all *mappings* $\mu|_{\vec{x}}$ such that μ is a homomorphism from $R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$ into $\mathcal{D}$, where $\mu|_{\vec{x}}$ denotes the restriction (or projection) of μ to the variables $\vec{x}$ (observe that usually, $q(\mathcal{D})$ is defined as the set of all tuples $\mu(\vec{x})$).

Graphs. We consider undirected, simple graphs $G = (V, E)$. For a graph G, we may write $V(G)$ and $E(G)$ to denote the set of nodes and edges, respectively. As usual, a *tree* is an acyclic graph, and a subtree is a connected subgraph of a tree. A *tree decomposition* of a graph $G = (V, E)$ is a pair (S, ν), where S is a tree and $\nu : V(S) \to 2^V$, that satisfies the following: (1) For each $u \in V$ the set $\{s \in V(S) \mid u \in \nu(s)\}$ is a connected subset of $V(S)$, and (2) each edge of E is contained in one of the sets $\nu(s)$, for $s \in V(S)$. The *width* of (S, ν) is $(\max\{|\nu(s)| \mid s \in V(S)\}) - 1$. The *treewidth* of G is the minimum width of its tree decompositions. Intuitively, the treewidth of G measures its *tree-likeness*. Notice that if G is an undirected graph, then G is acyclic iff it is of treewidth one.

Well-designed pattern trees. *Well-designed pattern trees* (wdPTs) were originally introduced in [23] as a graphical representation of *well-designed SPARQL* defined in [25] and later extended to arbitrary relational vocabulary [4]. Their formal definition is given below.

▶ **Definition 3** (wdPTs). A *well-designed pattern tree* (wdPT) p is a tuple $(T, \lambda, \vec{x})$, such that the following holds:

1. T is a rooted tree and λ maps each node $N \in V(T)$ to a set of relational atoms.
2. For every variable y mentioned in T, the set of nodes of T where y occurs is connected.
3. The tuple $\vec{x}$ of distinct variables from T denotes the *free variables* of the wdPT.

We say that $(T, \lambda, \vec{x})$ is *projection-free*, if $\vec{x}$ contains all variables mentioned in T.

Clearly, CQs correspond to the special case of wdPTs consisting of the root node only. Condition (2) above is referred to as the *well-designedness condition* introduced in [25]. We use upper-case letters to denote nodes of a wdPT, and lower-case letters for vertices of tree-decompositions and general graphs. For a wdPT $p = (T, \lambda, \vec{x})$ and a node $N \in V(T)$, we may abbreviate $var(\lambda(N))$ to $var(N)$. Also, for a subtree T' of T we may use $var(T')$ to denote the set $\bigcup_{N \in V(T')} var(N)$. Finally, we may write $var(p)$ instead of $var(T)$.

Let $p = (T, \lambda, \vec{x})$ be a wdPT. We write R to denote the root of T. Given a subtree T' of T rooted in R, we define $q_{T'}$ to be the CQ $\text{Ans}(\vec{y}) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$, where $\{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\} = \bigcup_{N \in T'} \lambda(N)$, and $\vec{y} = var(T')$. Finally, we write $|p|$ to denote the *size* of p in standard relational notation – which corresponds to the size of q_T.

We define the *semantics of wdPTs* by naturally extending their interpretation under semantic web vocabularies [23, 26]. Intuitively, a mapping μ satisfies (T, λ) over a database $\mathcal{D}$ if it is a solution to a CQ $q_{T'}$ and if no proper extension of μ is a solution to some CQ $q_{T''}$. The result of evaluating a wdPT $(T, \lambda, \vec{x})$ over $\mathcal{D}$ contains the projection of all mappings satisfying (T, λ) to $\vec{x}$. We formalize this next.

▶ **Definition 4** (Semantics of wdPTs). Let $p = (T, \lambda, \vec{x})$ be a wdPT and $\mathcal{D}$ a database.

- A *homomorphism* from p to $\mathcal{D}$ is a partial mapping μ for which there is a subtree T' of T rooted in R such that $\mu \in q_{T'}(\mathcal{D})$.
- A homomorphism μ is *maximal* if there is no homomorphism μ' from p to $\mathcal{D}$ s.t. $\mu \subset \mu'$.

The *evaluation* of wdPT $p = (T, \lambda, \vec{x})$ over $\mathcal{D}$, denoted $p(\mathcal{D})$, corresponds to the set of all mappings of the form $\mu|_{\vec{x}}$, such that μ is a maximal homomorphism from p to $\mathcal{D}$.

For a wdPT p and a database $\mathcal{D}$, a mapping μ is called a *partial solution* if there exists some solution $\mu' \in p(\mathcal{D})$ s.t. $\mu \subseteq \mu'$. Observe that $p(\mathcal{D})$ may contain mappings μ, μ' s.t. $\mu \subseteq \mu'$. As an example, recall the mappings μ'_1 and μ'_2 that are the result of restricting the mappings μ_1 and μ_2 from Example 2 to the variables y and z. We thus define the set of *maximal solutions* as $p_m(\mathcal{D}) = \{\mu \in p(\mathcal{D}) \mid \text{ for all } \mu' \in p(\mathcal{D})\colon \mu \not\subset \mu'\}$. For instance, in the above example, the maximal solutions contain only the mapping μ'_2.

Parameterized complexity. Let Σ be a finite alphabet. A *parameterization* of Σ^* is a polynomial time computable mapping $\kappa\colon \Sigma^* \to \mathbb{N}$. A *parameterized problem* over Σ is a pair (L, κ) where $L \subseteq \Sigma^*$ and κ is a parameterization of Σ^*. We refer to $x \in \Sigma^*$ as the instances of a problem, and to the numbers $\kappa(x)$ as the parameters. The following well-known problems will play an important role in our parameterized-complexity analyses.

p-Clique	
Instance:	A graph G and $k \in \mathbb{N}$.
Parameter:	k
Question:	Does G contain a clique of size k?

p-Dominating Set	
Instance:	A graph G and $k \in \mathbb{N}$.
Parameter:	k
Question:	Does G contain a dominating set of size k?

A parameterized problem $E = (L, \kappa)$ belongs to the class FPT of "fixed parameter tractable" problems if there exists an algorithm $\mathcal{A}$ deciding L, a polynomial p, and a computable function $f\colon \mathbb{N} \to \mathbb{N}$ such that the running time of $\mathcal{A}$ is at most $f(\kappa(x)) \cdot p(|x|)$.

Parameterized complexity theory also provides notions of intractability. Towards this notion, we first recall the definition of fpt-reductions. Let $E = (L, \kappa)$ and $E' = (L', \kappa')$ be parameterized problems over the alphabets Σ and Σ', respectively. An *fpt-reduction* from E to E' is a mapping $R\colon \Sigma^* \to (\Sigma')^*$ such that (1) for all $x \in \Sigma^*$ we have $x \in L$ iff $R(x) \in L'$, (2) there is a computable function f and a polynomial p such that $R(x)$ can be computed in time $f(\kappa(x)) \cdot p(|x|)$, and (3) there is a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

One notion of intractability for parameterized problems are the classes W[i] (for $i \geq 1$) of the W-hierarchy. Since we are here only interested in the classes W[1] and W[2], we omit a discussion of the W-hierarchy and only recall the following facts: A parameterized problem E is in W[1] or W[2], if there exists an fpt-reduction to the problems p-Clique (for W[1]) and p-Dominating Set (for W[2]), respectively. Similarly, E is W[1]-hard or W[2] if there exists an fpt-reduction from p-Clique and p-Dominating Set, respectively. It is strongly believed that problems that are hard for W[1] or W[2] are not in FPT. For details, see [13].

Complexity classes for enumeration problems. A *parameterized enumeration problem* is a triple $(L, \kappa, \mathsf{Sol})$ such that $L \subseteq \Sigma^*$ (for an alphabet Σ), κ is a parameterization of Σ^*, and $\mathsf{Sol}\colon \Sigma^* \to \mathcal{P}(\Sigma^*)$ is a function such that for all $x \in \Sigma^*$, we have that $\mathsf{Sol}(x)$ (the set of "solutions") is finite and $\mathsf{Sol}(x) = \emptyset$ iff $x \notin L$. When we omit the parameterization κ, the pair (L, Sol) is an *enumeration problem*.

An *enumeration algorithm* $\mathcal{A}$ for a (parameterized) enumeration problem $E = (L, \mathsf{Sol})$ (resp., $E = (L, \kappa, \mathsf{Sol})$) is an algorithm which, on input x, outputs exactly the elements from $\mathsf{Sol}(x)$ without duplicates. We denote the output of $\mathcal{A}$ on x by $\mathcal{A}(x)$.

Let $\mathcal{A}$ be an enumeration algorithm for some problem E. For an input x, let $n = |\mathcal{A}(x)|$. For $0 \leq i \leq n$, we define the *delay* $delay(i)$ as follows: $delay(0)$ ("preprocessing") is the time between the start of the algorithm and the (beginning of the) first output (or termination of

$\mathcal{A}$, if $n = 0$). For $0 < i < n$, $delay(i)$ is the time between outputting solution i and $(i+1)$. Finally, $delay(n)$ is the time between the last output and the termination of $\mathcal{A}$.

The concept of the delay between outputs allows us to define several classes of enumeration problems that capture different notions of tractability for these problems [17, 8, 31]. In the sequel, let $E = (L, \mathsf{Sol})$ (resp. $E' = (L, \kappa, \mathsf{Sol})$) be a (parameterized) enumeration problem. For a class $\mathcal{C}$, we say that $E \in \mathcal{C}$ ($E' \in \mathcal{C}$, respectively), if there exists an enumeration algorithm $\mathcal{A}$ for E, some $m \in \mathbb{N}$, and a computable function f such that on every input x the following holds:

- $E \in \mathsf{OutputP}$: $\mathcal{A}$ terminates in time $\mathcal{O}((|x| + |\mathsf{Sol}(x)|)^m)$.
- $E' \in \mathsf{OutputFPT}$: $\mathcal{A}$ terminates in time $\mathcal{O}(f(\kappa(x)) \cdot (|x| + |\mathsf{Sol}(x)|)^m)$.
- $E \in \mathsf{DelayP}$: $delay(i)$ is in $\mathcal{O}(|x|^m)$ for every $0 \leq i \leq |\mathsf{Sol}(x)|$.
- $E' \in \mathsf{DelayFPT}$: $delay(i)$ is in $\mathcal{O}(f(\kappa(x)) \cdot |x|^m)$ for every $0 \leq i \leq |\mathsf{Sol}(x)|$.
- $E \in \mathsf{DelayC_{lin}}$: $delay(0)$ is in $\mathcal{O}(|x|)$ and for $0 < i \leq |\mathsf{Sol}(x)|$, $delay(i)$ is in $\mathcal{O}(1)$.

A stricter notion of allowing for polynomial delay between two solutions is captured by the class $\mathsf{SDelayP}$ ("strict polynomial delay"): An enumeration problem E is in $\mathsf{SDelayP}$ if there exists a total order $<$ on $\mathsf{Sol}(x)$, some $m \in \mathbb{N}$, and an algorithm $\mathcal{B}$ terminating in time $\mathcal{O}(|x|^m)$ with the following output: On input x, it returns the first element of $\mathsf{Sol}(x)$ (according to $<$). On input (x, y) for any $y \in \mathsf{Sol}(x)$, it either returns the next element according to $<$ if there is some, or halts otherwise. As is common in the literature on enumeration, we use RAMs (rather than Turing Machines) as model of computation [17].

3 Parameterized Complexity of Evaluation

While there has been some research on the classical complexity of fragments of wdPTs [4], the parameterized complexity is still open. We thus recall two variants of the evaluation problem, namely Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$) (where we ask if a given mapping is a solution or a maximal solution, respectively) and introduce the parameterized versions p-Eval($\mathcal{C}$) and p-Max-Eval($\mathcal{C}$). Here, the size of the query is taken as parameter.

Eval($\mathcal{C}$) / Max-Eval($\mathcal{C}$)	
Instance:	Query: wdPT $p \in \mathcal{C}$, mapping μ, Data: database $\mathcal{D}$.
Question:	$\mu \in p(\mathcal{D})$ / $\mu \in p_m(\mathcal{D})$?

p-Eval($\mathcal{C}$) / p-Max-Eval($\mathcal{C}$)	
Instance:	wdPT $p \in \mathcal{C}$, mapping μ, database $\mathcal{D}$.
Paramter:	$\lvert p \rvert$
Question:	$\mu \in p(\mathcal{D})$ / $\mu \in p_m(\mathcal{D})$?

Observe that in order to talk about the common notions of data- and query complexity, for Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$) we distinguish which parts of the input are considered to be part of the query, and which are part of the data. For the parameterized case, because of the explicit parameter, such a distinction does not make sense.

Tractable CQ-evaluation. When looking for tractable classes of wdPT evaluation [4], a natural starting point are tractable classes of CQ evaluation. We define the problem CQ-Eval($\mathcal{C}$) for a class $\mathcal{C}$ of CQs as follows: Given a CQ $q \in \mathcal{C}$, a mapping μ, and a database $\mathcal{D}$ as input, decide if $\mu \in q(\mathcal{D})$. This problem has been extensively studied and several tractable classes of CQs have been identified [32, 14, 15]. If the maximal arity of the relation symbols is known in advance, even the precise border of tractability is known [16].

An important tractable class of CQs is obtained by restricting the *treewidth* of the *Gaifman-graph* of queries [6]. The Gaifman-graph of a CQ q is a graph $G = (V, E)$ where $V = var(q)$ and E contains exactly all pairs of variables that jointly occur in some atom

in q. The treewidth of a CQ is the treewidth of its Gaifman-graph. We denote by $\mathsf{TW}(k)$ the class of CQs of treewidth at most k. It follows from [6] (see also [10]) that for $k \geq 1$, CQ-Eval($\mathsf{TW}(k)$) is in PTIME.

Tractable wdPT-evaluation. In the past, tractable classes of CQs have been successfully used for defining tractable classes of wdPTs following two different approaches. Either by locally restricting the CQs defined by each node of the wdPT to be from some tractable class (*"local tractability"*), or by globally requiring that for every subtree (containing the root) the corresponding CQ is tractable (*"global tractability"*). Formally, let $\mathcal{C}$ be a class of CQs. A wdPT $p = (T, \lambda, \vec{x})$ is *locally in* $\mathcal{C}$ if for every node $N \in V(T)$ the CQ $\text{Ans}() \leftarrow \lambda(N)$ is in $\mathcal{C}$. Also, p is *globally in* $\mathcal{C}$ if for every subtree T' of T rooted in R the CQ $q_{T'}$ is in $\mathcal{C}$. We denote with ℓ-$\mathcal{C}$ and g-$\mathcal{C}$ the sets of all wdPTs that are locally and globally in $\mathcal{C}$, respectively.

It was shown for projection-free wdPTs that local tractability is sufficient to achieve tractability. However, in the presence of projection, Eval(ℓ-$\mathsf{TW}(k)$) and Max-Eval(ℓ-$\mathsf{TW}(k)$) are NP-complete [23] and DP-complete [4], respectively. Resorting to global tractability only helps when dealing with maximal solutions: Max-Eval(g-$\mathsf{TW}(k)$) is in PTIME, while Eval(g-$\mathsf{TW}(k)$) remains NP-complete [4].

Thus, just restricting the structure of the CQs defined by a wdPT is not sufficient for tractability. Instead, an additional source of complexity comes from the information shared between different nodes. Thus another approach is to restrict the number of variables nodes may have in common. Formally, let $p = (T, \lambda, \vec{x})$ be a wdPT. For two nodes $N, M \in V(T)$, we define the interface $\mathcal{I}(N, M) = var(N) \cap var(M)$. Similarly, for a node $N \in V(T)$ define $\mathcal{I}(N) = \bigcup_{M \in (V(T) \setminus \{N\})} \mathcal{I}(N, M)$. Observe that because of condition (2) in the definition of wdPTs, the interface of a node N can be completely determined by just looking at its neighbors, i.e. for every $N \in V(T)$ we have $\bigcup_{M \in (V(T) \setminus \{N\})} \mathcal{I}(N, M) = \bigcup_{\{M \mid \{N,M\} \in E(T)\}} \mathcal{I}(N, M)$. For $c \geq 0$, we say that p has *c-bounded interface* if $|\mathcal{I}(N)| \leq c$ for all $N \in V(T)$. Similarly, we say that p has *c-semi-bounded interface* if for any two distinct nodes $N, M \in V(T)$ we have $|\mathcal{I}(N, M)| \leq c$. We denote with $\mathsf{BI}(c)$ and $\mathsf{SBI}(c)$ the classes of wdPTs of c-bounded interface and c-semi-bounded interface, respectively.

Of course, restricting only the number of shared variables is not sufficient for tractability, since already the evaluation of the CQ in the root node is intractable. However, combining the two approaches above finally leads to tractable classes of wdPTs: Let $\mathcal{C}$ be a class of CQs s.t. CQ-Eval($\mathcal{C}$) is in PTIME and $c \geq 1$. Then Eval(ℓ-$\mathcal{C} \cap \mathsf{BI}(c)$) is also in PTIME [4].

An inspection of the NP-hardness proof for Eval(g-$\mathsf{TW}(k)$) provided in [4] reveals that hardness even holds for Eval(g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$), but this class is newly introduced in the present paper and was not considered in [4]. The introduction of the notion of a semi-bounded interface allows us to define the classes ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$ and g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$. Below, we show that a parameterized complexity analysis of these classes helps to explore the gap between tractability of Eval(ℓ-$\mathcal{C} \cap \mathsf{BI}(c)$) and the NP-completeness of Eval(g-$\mathsf{TW}(k)$). In fact, we show that the parameterized evaluation problem for the different classes of wdPTs is in PTIME (for ℓ-$\mathcal{C} \cap \mathsf{BI}(c)$), in FPT (for g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$), W[1]-complete (for ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$), and W[2]-hard (for g-$\mathsf{TW}(k)$), respectively.

Evaluation in case of bounded vs. semi-bounded interface. Before we present our FPT-result for wdPTs in g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$, we first discuss the main difference compared with the restriction to ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$. In the latter case, it is easy to construct a global tree decomposition of width $k + c$, s.t. for any two neighboring nodes N, M in the wdPT, the interface $\mathcal{I}(N, M)$ is covered by some bag in the tree decomposition. Indeed, we can take a

local tree decomposition for every node N (of width at most k) and add all interface variables to every bag. A global tree decomposition of p is then obtained by gluing together the local tree decompositions. With this global tree decomposition, the FPT-membership (actually, even the PTIME-membership) is easily established [4]. In contrast, for semi-bounded interface, the variables in $\mathcal{I}(N, M)$ for neighboring nodes N, M in p can be arbitrarily scattered over the global tree decomposition. Since the total number of interface variables of a single node in p with all its neighbors is unbounded, we cannot apply the above trick to construct a tree decomposition where all interfaces are covered by some bag. Hence, a different approach is needed in the following theorem to obtain FPT-membership also in this case.

We first introduce the decision problem ExtendSolution($\mathcal{C}$) (a similar problem was introduced in [7, 9] to study enumeration problems), defined as follows: given a database $\mathcal{D}$, a wdPT $p \in \mathcal{C}$, a partial mapping μ and a set $\vec{x}' \subseteq \vec{x}$ (where $\vec{x}$ are the free variables of p), does there exist some mapping $\mu' \in p(\mathcal{D})$ such that $\mu \subseteq \mu'$ and $dom(\mu') \cap \vec{x}' = \emptyset$? Observe that the last property $dom(\mu') \cap \vec{x}' = \emptyset$ is essential, since it allows us to explicitly specify variables $\vec{x}'$ which we do not want to be bound by the desired solutions. Clearly, the problem Eval($\mathcal{C}$) corresponds to the special case of the ExtendSolution($\mathcal{C}$) problem where we set $\vec{x}' \cup dom(\mu) = \vec{x}$, i.e., we ask if μ itself is the desired mapping μ'.

▶ **Theorem 5.** *The problem* p-Eval(g-TW(k) ∩ SBI(c)) *is in* FPT *for every* $k, c \geq 1$.

Proof idea. We prove FPT-membership for the ExtendSolution(g-TW(k) ∩ SBI(c)) problem, from which the desired FPT-result follows. Let p be a wdPT and $\mathcal{D}$ a database. Further let $\mathcal{T}$ denote a global tree decomposition of p of width k, let $\vec{x}$ denote the set of free variables in p, and let μ be a mapping with $dom(\mu) \subseteq \vec{x}$. Let $\mathcal{N}$ denote a set of nodes in p with $dom(\mu) \subseteq var(\mathcal{N})$ and let $\mathcal{M} = \{M_1, \ldots, M_\beta\}$ denote the set of nodes outside $\mathcal{N}$ whose parent is in $\mathcal{N}$. We have to test if there exists an extension ν of μ on the existentially quantified variables in $\mathcal{N}$, s.t. ν cannot be further extended to any of the nodes M_i in $\mathcal{M}$.

The key idea is, for all $M_i \in \mathcal{M}$, to define *critical subsets* $\mathcal{C}(N_i, M_i)$ of $\mathcal{I}(N_i, M_i)$, where $N_i \in \mathcal{N}$ is the parent of M_i. Intuitively, $\mathcal{C}(N_i, M_i)$ is defined in such a way that the existence of an extension ν of μ to the variables in M_i only depends on the values for μ on each of the critical subsets. Our FPT-algorithm relies on several crucial properties of $\mathcal{C}(N_i, M_i)$:

First of all, for each critical subset $\vec{v} \subseteq \mathcal{I}(N_i, M_i)$, we can efficiently determine the "good" (resp. "bad") value combinations, i.e., value combinations such that an extension (resp. no extension) of a mapping ν to M_i is possible, namely: $good(\vec{v}) = \{\eta \mid dom(\eta) = \vec{v}$ and there exists an extension ν of η to $var(M_i)$ with $\nu(M_i) \subseteq \mathcal{D}\}$ and $bad(\vec{v}) = \{\eta \mid dom(\eta) = \vec{v}$ and there exists no extension ν of η to $var(M_i)$ with $\nu(M_i) \subseteq \mathcal{D}\}$. It can be shown that, for an arbitrary mapping μ with $\mathcal{I}(M_i, N_i) \subseteq dom(\mu)$, there exists an extension ν of μ with $var(M_i) \subseteq dom(\nu)$ and $\nu(M_i) \subseteq \mathcal{D}$ iff for every $\vec{v} \in \mathcal{C}(M_i, N_i)$, we have $\mu_{|\vec{v}} \in good(\vec{v})$.

Consider μ from our arbitrary instance of ExtendSolution(g-TW(k) ∩ SBI(c)). We have to test if there exists an extension ν of μ to the existentially quantified variables in $\mathcal{N}$, s.t. ν cannot be further extended to any of the nodes M_i in $\mathcal{M}$. In other words, such ν has to satisfy the following two conditions: (1) $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}$ and (2) for every $i \in \{1, \ldots, \beta\}$, there exists a critical subset $\vec{v}_i \in \mathcal{C}(M_i, N_i)$, s.t. $\nu_{|\vec{v}_i} \in bad(\vec{v}_i)$.

Hence, our decision procedure for ExtendSolution(g-TW(k)∩SBI(c)) just checks if such a combination $(\vec{v}_1, \ldots, \vec{v}_\beta)$ of critical subsets exists. We can search for such a combination by nested loops over all $\vec{v}_i \in \mathcal{C}(M_i, N_i)$ with $i \in \{1, \ldots, \beta\}$. Since $\vec{v}_i \subseteq \mathcal{I}(M_i, N_i)$ and $|\mathcal{I}(M_i, N_i)| \leq c$, there are at most 2^c elements in each $\mathcal{C}(M_i, N_i)$. Moreover, β is bounded by the size of p. Hence, we have to check at most $f(p) = (2^c)^{|p|}$ combinations $(\vec{v}_1, \ldots, \vec{v}_\beta)$. To prove the algorithm to be in FPT, it suffices to show that, for a given combination $(\vec{v}_1, \ldots, \vec{v}_\beta)$

of critical subsets, one can test in polynomial time if there exists an extension ν of μ with (1) $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}$ and (2) $\nu_{|\vec{v}_i} \in bad(\vec{v}_i)$ for every $i \in \{1, \ldots, \beta\}$.

The second crucial property of the critical subsets $\mathcal{C}(N_i, M_i)$ with $i \in \{1, \ldots, \beta\}$ is that for any combination $(\vec{v}_1, \ldots, \vec{v}_\beta)$ with $\vec{v}_i \in \mathcal{C}(N_i, M_i)$, we can transform the given global tree decomposition $\mathcal{T}$ of p of width k into a tree decomposition $\mathcal{T}^*$ of width $\leq (k+1) \cdot (c+1)$, s.t. every $\vec{v}_i$ is covered by the bag of some vertex t in $\mathcal{T}^*$. Analogously to the PTIME-membership proof in [4], the tree decomposition $\mathcal{T}^*$ guarantees that the check for the existence of an extension ν of μ with the desired properties (1) and (2) is feasible in polynomial time. ◀

What happens if, instead of the restriction to g-TW$(k) \cap$ SBI(c), we consider ℓ-TW$(k) \cap$ SBI(c)? Below we show that, with this relaxation, fixed-parameter tractability is lost.

▶ **Theorem 6.** *The problem* p-Eval(ℓ-TW$(k) \cap$ SBI(c)) *is* W[1]*-complete for* $k \geq 1$ *and* $c \geq 2$.

Proof sketch. *Membership* is shown by reduction to the W[1]-complete problem Pos-Eval, which is the evaluation problem for FO-queries built from relational atoms using $\exists, \wedge, \vee$ [24] (see also [24] for the problem definition). The idea is to create one positive query which is a big disjunction over all possible CQs $q_{T'}$ corresponding to subtrees of p that potentially have μ as an answer. The maximality of answers is enforced by introducing "interface relations" which, for every child node, only contain those tuples which cannot be extended to the child.

Hardness is shown by an fpt-reduction from p-Clique. Given a graph $G = (V, E)$ and $d \in \mathbb{N}$, we construct a wdPT p and a database $\mathcal{D}$ with the following intuition: The root R of p contains d variables $x_1, \ldots, x_d$. Mapping the root into $\mathcal{D}$ assigns one node from V to each x_i. In addition, R contains one child for each pair of distinct variables $x_\alpha, x_\beta \in \{x_1, \ldots, x_d\}$. Each of these children can be mapped into $\mathcal{D}$ iff there is an edge between the nodes assigned to x_α, x_β. Thus G contains a clique of size d iff there exists a mapping $\mu \in p(\mathcal{D})$ that maps all children of R into $\mathcal{D}$. ◀

We now consider another relaxation of the g-TW$(k) \cap$ SBI(c) restriction from Theorem 5 by considering wdPTs in g-TW(k) but without any bound on the interfaces.

▶ **Theorem 7.** *The problem* p-Eval(g-TW(k)) *is* W[2]*-hard for* $k \geq 1$.

Proof sketch. The proof is by reduction from p-Dominating Set. Thus, let $G = (V, E)$ be a graph and $d \in \mathbb{N}$. The idea of the constructed wdPT p and database $\mathcal{D}$ is to have variables $x_1, \ldots, x_d$ in the root R of p such that a mapping of R into $\mathcal{D}$ assigns one node from V to each x_i. Observe that $S \subseteq V$ is a dominating set iff there does not exist some $u \in V \setminus S$ that is not adjacent to any node in S. This is tested in the single child node of R: It contains an additional variable x_0 which also encodes nodes in V. Now the child node is mapped into $\mathcal{D}$ iff there exists a value for x_0 that differs from those of all x_i's, and there does not exist an edge between the nodes mapped to x_0 and any of the x_i's. I.e., there exists a solution that only maps the root into $\mathcal{D}$ iff G contains a dominating set of size d. ◀

It was shown in [4] that the problem Max-Eval(g-TW(k)) is in PTIME. In other words, for wdPTs with globally bounded treewidth, there is no need to also restrict the interface. Moreover, as recalled above, restricting wdPTs to ℓ-TW$(k) \cap$ BI(c) also yields a restriction of the global treewidth. The only case remaining is therefore the restriction to ℓ-TW$(k) \cap$ SBI(c).

▶ **Proposition 8.** *The problem* p-Max-Eval(ℓ-TW$(k) \cap$ SBI(c)) *is* W[1]*-hard for* $k \geq 1$ *and* $c \geq 2$.

Proof. The reduction used to prove the hardness in the proof of Theorem 6 also proves this case, since clearly $\mu \in p(\mathcal{D})$ iff $\mu \in p_m(\mathcal{D})$. ◀

4 Classical and Parameterized Complexity of Enumeration

We now turn our attention to the enumeration problem. Analogously to the evaluation problem in Section 3, we will study four variants of the enumeration problem, namely enumerating *all* vs. the *maximal* solutions and *parameterized* vs. *non-parameterized* problems.

Enum($\mathcal{C}$) / Max-Enum($\mathcal{C}$)	
Instance:	Query: wdPT $p \in \mathcal{C}$. Data: database $\mathcal{D}$.
Output:	$p(\mathcal{D})$ / $p_m(\mathcal{D})$?

p-Enum($\mathcal{C}$) / p-Max-Enum($\mathcal{C}$)	
Instance:	wdPT $p \in \mathcal{C}$, database $\mathcal{D}$.
Parameter:	$\lvert p\rvert$
Output:	$p(\mathcal{D})$ / $p_m(\mathcal{D})$?

For CQs, the enumeration problem is also well-studied, although not as thoroughly as the evaluation problem. Similarly to the evaluation problem, restrictions on the graph structure of the query have proven useful when looking for tractable enumeration of CQs, cf. [32, 5, 3]. For wdPTs, analogously to the evaluation problem, additional restrictions are necessary in order to achieve tractability. However, for enumeration we get a more diverse picture than for evaluation: When we are interested in all solutions, the additional restrictions used for evaluation are sufficient also for enumeration, while this is not the case if we are only interested in the maximal solutions. It will turn out that the techniques required to analyze the enumeration of *all* vs. the *maximal* solutions differ significantly. We thus treat the enumeration and the max-enumeration problems in separate subsections below.

4.1 Enumeration

For our study of the enumeration problem, the decision problem ExtendSolution($\mathcal{C}$), which we introduced in Section 3, again plays an important role.

▶ **Lemma 9.** *Let $\mathcal{C}$ be a class of pattern trees, $p = (T, \lambda, \vec{x}) \in \mathcal{C}$, and $\mathcal{D}$ a database. Assume that there is a computable function f such that for every partial mapping μ and every subset $\vec{x}'$ of $\vec{x}$, the problem* ExtendSolution($\mathcal{C}$) *can be decided in $\mathcal{O}(f(|p|, |\mathcal{D}|))$. Then there exists an algorithm enumerating $p(\mathcal{D})$ with delay(i) in $\mathcal{O}(f(|p|, |\mathcal{D}|) \cdot |\mathcal{D}| \cdot |p|)$ for $i \geq 0$.*

Proof Sketch. The general idea is to create solutions by iteratively testing if certain variable bindings can be extended to solutions. Observe that for CQs, it would suffice to test (in nested loops) for each variable if it can be bound to a certain domain element. In contrast, for wdPTs, we now have to consider an additional option, namely not binding a variable at all. Thus, we look for extensions (= solutions) that bind some variables to specific domain values (expressed by μ) and leave some variables unbound (expressed by $\vec{x}'$).

In a little bit more detail, we enumerate all $\nu \in p(\mathcal{D})$ as follows: For all $a \in dom(\mathcal{D})$, we can check using ExtendSolution($\mathcal{C}$) whether $\{(x_1, a)\}$ can be extended to a solution (by setting $\mu = \{(x_1, a)\}$), and whether there is a solution $\nu \in p(\mathcal{D})$ with $x_1 \notin dom(\nu)$ (by setting $\vec{x}' = \{x_1\}$). Then by either fixing such a value $a_1 \in dom(\mathcal{D})$, or by fixing the fact that x_1 will not be in the domain of an extension, we can repeat this test for all $a_2 \in dom(\mathcal{D})$. Thus by iteratively fixing such variable assignments (respectively intentional non-assignments) for all n variables x_i in $\vec{x}$ by either extending μ or $\vec{x}'$, we can output a mapping $\rho \in p(\mathcal{D})$ in the n-th step. Given a solution $\rho \in p(\mathcal{D})$, we find the next one by taking the maximal $j \in \{1, \ldots, n\}$ such that under the same assignments for $x_1, \ldots, x_j$, the check for an extension is positive for a different $\nu(x_j)$. Extending this assignment as described above gives a $\rho' \neq \rho$ with $\rho' \in p(\mathcal{D})$. Iterating this process outputs $p(\mathcal{D})$ with a delay of $\mathcal{O}(f(|p|, |\mathcal{D}|) \cdot |\mathcal{D}| \cdot |p|)$. ◀

To make use of this lemma, we identify classes of wdPTs that meet the requirements:

▶ **Proposition 10.** *The following complexity results hold for* ExtendSolution($\mathcal{C}$)*:*
1. *Let $k, c \geq 1$ and $\mathcal{C}$ be a class of CQs for which* CQ-Eval($\mathcal{C}$) *is in* PTIME*. Then* ExtendSolution(ℓ-$\mathcal{C} \cap$ BI(c)) *is in* PTIME*.*
2. ExtendSolution(g-TW(k)) *is* NP*-complete for every $k \geq 1$.*
3. *Let $k, c \geq 1$. Then* ExtendSolution(g-TW(k) $\cap$ SBI(c)) *parameterized by $|p|$ is in* FPT*.*

Proof idea. (1) follows by some slight modifications of the proof of [4, Theorem 6]. Since Eval($\mathcal{C}$) is a special case of ExtendSolution($\mathcal{C}$) where $\vec{x}' = \vec{x} \setminus dom(\mu)$, (2) follows immediately from [4]. For (3) we note that in the previous section, Theorem 5 was stated for p-Eval(g-TW(k) $\cap$ SBI(c)). One can actually show the stronger result of FPT membership for the parameterized version of ExtendSolution(g-TW(k) $\cap$ SBI(c)). ◀

Now the following corollary follows immediately from the previous results.

▶ **Corollary 11.** *Let $k, c \geq 1$.*
- *Let $\mathcal{C}$ be a class of CQs for which* CQ-Eval($\mathcal{C}$) *is in* PTIME*. Then* Enum(ℓ-$\mathcal{C} \cap$ BI(c)) *is in* SDelayP*.*
- *The problem p-*Enum(g-TW(k) $\cap$ SBI(c)) *is in* DelayFPT*.*

We now move to negative results for the enumeration problem. As an important tool and an interesting result in its own right, we establish a close relationship between enumeration and the parameterized complexity of evaluation. We introduce some terminology first:

▶ **Definition 12.** A class $\mathcal{C}$ of wdPTs is *robust* if for every wdPT $p = (T, \lambda, \vec{x}) \in \mathcal{C}$ the following two conditions hold:
1. For every $\mathcal{N} = \{N_1, \ldots, N_m\} \subseteq V(T)$ the wdPT $(T, \lambda_{\mathcal{N}}, \vec{z})$ is in $\mathcal{C}$, where $\vec{z} = \{z_1 \ldots, z_m\}$ is a set of new variables, $\lambda_{\mathcal{N}}(N) = \lambda(N)$ for all $N \in V(T) \setminus \mathcal{N}$ and $\lambda_{\mathcal{N}}(N_i) = \lambda(N_i) \cup \{b(z_i)\}$ for $1 \leq i \leq m$ and some new relation symbol b.
2. For every variable $x \in var(p)$, let p' be the wdPT retrieved from p by replacing every occurrence of x by the same constant c. Then $p' \in \mathcal{C}$.

The notion of "robust" classes of wdPTs is important in the following theorem, to make sure that wdPTs do not fall out of their class when certain transformations are performed.

▶ **Theorem 13.** *Let $\mathcal{C}$ be a robust class of wdPTs. If p-*Enum($\mathcal{C}$) *is in* OutputFPT*, then p-*Eval($\mathcal{C}$) *is in* FPT*.*

Proof Sketch. Given a pattern tree $p = (T, \lambda, \vec{x})$, a database $\mathcal{D}$ and a mapping μ, we first transform p into a pattern tree p' by substituting free variables in p according to μ, and then adding unary atoms of the form $b(z_i)$ with new variables to the leaf nodes of the minimal subtree of T only containing the variables $dom(\mu)$ and to the children of the leaf nodes of the maximal subtree containing only $dom(\mu)$. Further we set the free variables of p' to the newly introduced variables and extend the database to a database $\mathcal{D}' = \mathcal{D} \cup \{b(1)\}$. We can fix a mapping $\mu' \in p'(\mathcal{D}')$ such that $\mu \in p(\mathcal{D})$ iff $\mu' \in p'(\mathcal{D}')$. Since $|p'(\mathcal{D}')|$ is in $\mathcal{O}(|p| \cdot 2^{|p|})$, we can output all of $p'(\mathcal{D}')$ in $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^m)$ for some $m \geq 1$ and some function f, and hence decide whether $\mu' \in p'(\mathcal{D}')$ and thus whether $\mu \in p(\mathcal{D})$ in time in $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^m)$. ◀

Exploiting this relationship between the evaluation and the enumeration problem, the next results are immediate consequences of the W[1]- and W[2]-hardness, respectively, shown in the previous section and the fact that all the classes mentioned in this paper are robust.

▶ **Corollary 14.** *If* FPT $\neq$ W[1]*, then the following holds:*

- *The problem p-*Enum(ℓ-TW(k) $\cap$ SBI(c)) *is not in* OutputFPT *for* $k \geq 1$ *and* $c \geq 2$*. Thus also the problem* Enum(ℓ-TW(k) $\cap$ SBI(c)) *is not in* OutputP *for* $k \geq 1$ *and* $c \geq 2$*.*
- *The problem p-*Enum(g-TW(k)) *is not in* OutputFPT *for* $k \geq 1$*.*

It follows of course also immediately from Theorem 13 (in combination with Theorem 7) that Enum(g-TW(k)) is not in OutputP if FPT $\neq$ W[1]. However, we can show the same result under an even stronger assumption, namely assuming that PTIME $\neq$ NP, by making use of the following NP-hardness result.

▶ **Proposition 15.** *The following problem is* NP*-hard for every* $k \geq 1$*: Given a wdPT* $p \in g$-TW(k) *and a database* $\mathcal{D}$*, decide whether* $|p(\mathcal{D})| = 2$*.*

Proof sketch. The proof is by reduction from the NP-complete problem Dominating Set, via the same reduction used to prove Theorem 7. W.l.o.g. we consider only graphs G that contain at least one node that is not already a dominating set. Then there always exists one solution that maps both, the root and the child into $\mathcal{D}$. In addition there still exists a second solution that maps only the root into $\mathcal{D}$ iff G contains a dominating set of size d. ◀

The next result follows immediately. Indeed, an algorithm solving Enum(g-TW(k)) in polynomial time w.r.t. the size of the input plus the output would provide a polynomial time decision procedure for Dominating Set, a problem well-known to be NP-hard.

▶ **Corollary 16.** *If* PTIME $\neq$ NP*, then* Enum(g-TW(k)) *is not in* OutputP *for every* $k \geq 1$*.*

4.2 Max-Enumeration

We next turn our attention to the problem of enumerating only the *maximal solutions* of a wdPT. In fact, we show that for none of the classes of wdPTs considered in this work, the problem Max-Enum(ℓ-TW(k) $\cap$ BI(c)) is in OutputP. After establishing this result, we therefore turn towards the parameterized problem, where for all but one of the classes we can show a positive result, namely DelayFPT membership.

Of course, for the negative result on the non-parameterized problem, it suffices to show intractability for Max-Enum(ℓ-TW(k) $\cap$ BI(c)), since the other results follow from that. Again, we do this via an intractability result for a suitable decision problem.

▶ **Proposition 17.** *The following problem is* NP*-hard for every* $k, c \geq 1$*: Given a wdPT* $p \in \ell$-TW(k) $\cap$ BI(c)*, a database* $\mathcal{D}$*, and an integer* $s \geq 1$ *encoded in unary, decide if* $|p_m(\mathcal{D})| > s$*.*

To show that Max-Enum(ℓ-TW(k) $\cap$ BI(c)) is not in OutputP using this result, we recall a relationship from [31] (also [18]). The lemma below is actually a slight reformulation of [31, Lemma 2.11]. However, it holds by the same arguments as the original statement.

▶ **Lemma 18** ([31])**.** *Let* E *be an enumeration problem. If* E *is in* OutputP*, then the following problem is in* PTIME*: Given an instance* x *of* E *and an integer* s *encoded in unary, decide if* $|E(x)| > s$*.*

The intractability of Max-Enum(ℓ-TW(k) $\cap$ BI(c)) now is an immediate consequence of the previous two results.

▶ **Corollary 19.** *If* PTIME $\neq$ NP*, then* Max-Enum(ℓ-TW(k) $\cap$ BI(c)) *is not in* OutputP *for every* $k, c \geq 1$*.*

We now show that the parameterized problem p-Max-Enum($\mathcal{C}$) is tractable in all but one cases. We first establish the tractability for $\mathcal{C} = g\text{-}\mathsf{TW}(k)$ and $\mathcal{C} = \ell\text{-}\mathsf{TW}(k) \cap \mathsf{BI}(c)$, respectively. Of course, this immediately implies tractability for $\mathcal{C} = g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$.

We start by formulating a crucial lemma. It will be convenient to recall the problem Partial-Eval($\mathcal{C}$) from the literature (cf. [4]): Given a wdPT p, a database $\mathcal{D}$ and a mapping μ, does there exist some $\mu' \in p(\mathcal{D})$ s.t. $\mu \subseteq \mu'$?

▶ **Lemma 20.** *Let $\mathcal{C}$ be a class of wdPTs such that* Partial-Eval($\mathcal{C}$) *is in* PTIME. *Assume that there exists an enumeration algorithm $\mathcal{A}$, such that for every $p \in \mathcal{C}$ and every database $\mathcal{D}$, $\mathcal{A}$ enumerates some set P of partial solutions with $p_m(\mathcal{D}) \subseteq P$ with delay(i) in $\mathcal{O}(f(|p|)\cdot|\mathcal{D}|^m)$ for some computable function f, some positive integer m and $0 \leq i \leq |P|$. Then p-Max-Enum($\mathcal{C}$) is in* DelayFPT.

Proof idea. Given the enumeration algorithm $\mathcal{A}$ for P, the idea is to construct an extension $\mathcal{A}'$ of $\mathcal{A}$ that, after initializing a set $M = \emptyset$, performs the following steps: (1) Retrieve the next output μ of $\mathcal{A}$ and extend it to a maximal solution μ_m. If μ_m has not been created before, add μ_m to M; (2) If the previous step was repeated $2^{|p|}$ times since the last output, output and delete a mapping from M. If $\mathcal{A}$ halts, $\mathcal{A}'$ outputs M and halts as well.

Two main observations are important: First, if Partial-Eval($\mathcal{C}$) is in PTIME, we can always extend a (partial) solution to a maximal solution by greedily adding variable mappings. Second, at most $2^{|p|}$ partial solutions can be extended to the same maximal solution. ◀

For a class $\mathcal{C}$ of wdPTs to show that Max-Enum($\mathcal{C}$) is in DelayFPT it thus remains to identify a suitable set of partial solutions. We do this for the classes mentioned before.

▶ **Theorem 21.** *Let $\mathcal{C}'$ be a class of CQs s.t.* CQ-Eval($\mathcal{C}'$) *is in* PTIME, *$k, c \geq 1$, and $\mathcal{C} \in \{g\text{-}\mathcal{C}', l\text{-}\mathcal{C}' \cap \mathsf{BI}(c)\}$. Then the problem p-Max-Enum($\mathcal{C}$) is in* DelayFPT.

Proof idea. If CQ-Eval($\mathcal{C}'$) is in PTIME, then the corresponding enumeration problem is in DelayP (cf. [5]). Thus, for wdPTs $p \in g\text{-}\mathcal{C}'$, we have $q_{T'} \in \mathcal{C}'$ for every subtree T' of T containing the root. Thus $\bigcup_{T'} q_{T'}(\mathcal{D})$ gives the required set of partial solutions – it suffices to compute the solutions for one CQ after the other. For $l\text{-}\mathcal{C}' \cap \mathsf{BI}(c)$, we use $p(\mathcal{D})$ which can be efficiently enumerated by Corollary 11. ◀

Algorithm $\mathcal{A}'$ sketched in the proof of Lemma 20 crucially depends on the choice of RAMs as the model of computation: $\mathcal{A}'$ may need to store an exponential number of maximal solutions. A Turing Machine (TM) cannot access these solutions efficiently, while a RAM can. However, these algorithms could be easily adapted to run in *incremental delay* on a TM, i.e. for some $m \in \mathbb{N}$ and computable function f, *delay*(i) is in $\mathcal{O}(f(|p|) \cdot (|p| + |\mathcal{D}| + \sum_{j=1}^{i} |y_i|)^m)$ for $i \geq 0$ (where $y_1, \ldots, y_i$ are the first i solutions returned by the algorithm).

We have thus shown fixed-parameter tractability for three out of the four classes we consider. We conclude this section by showing that for the remaining class the problem is not in OutputFPT.

▶ **Proposition 22.** *If* FPT $\neq$ W[1], *then p-*Max-Enum($\ell\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$) *is not in* OutputFPT *for every $k, c \geq 1$.*

Proof idea. The proof is based on the same parameterized reduction from p-Clique used in the proof of Theorem 6. There we have that $|p_m(\mathcal{D})| \leq 2^{|p|}$. Thus, if enumerating $p_m(\mathcal{D})$ were in OutputFPT, then deciding p-Clique would be in FPT. ◀

5 Constant Delay

We now move to the data complexity of the enumeration problem. To make this explicit in the notation used below, we write ENUM_p to denote the problem $\text{ENUM}(\mathcal{C})$ where the query is considered fixed and the input consists of the database only. The usual goal in the database context is to devise an enumeration algorithm that works with linear time preprocessing and then outputs the solutions with constant delay [29, 30]. To reach this goal, typically additional restrictions on the query (and/or the data) are required than just acyclicity/bounded treewidth [12, 19, 3]. For instance, for CQs it was shown that under reasonable complexity assumptions, the class of conjunctive queries that are *free-connex acyclic* (respectively of *bounded free-connex treewidth*) is the only class of acyclic (bounded treewidth) CQs which allows constant delay enumeration [3].

Below, we show that this goal is not reachable for the classes of wdPTs studied here. Recall from Section 4 that lower bounds are proved relative to some common complexity theoretic assumption such as $\mathsf{PTIME} \neq \mathsf{NP}$ or $\mathsf{FPT} \neq \mathsf{W}[1]$. The assumptions used in this section are of a different nature, namely assuming that certain upper bounds are not achievable for the search of a triangle in a graph [21] and for the multiplication of Boolean matrices [22]. Observe that these are typical problems for showing these kind of lower bounds and have been used e.g. in [3] to show the dichotomy result mentioned above.

We first revisit the case of wdPTs without projections, i.e. let $\mathcal{C}_{\mathsf{pf}}$ be the class of wdPTs $p = (T, \lambda, \vec{x})$ such that $\vec{x} = var(p)$.

▶ **Proposition 23.** *If it is not possible to decide in time $\mathcal{O}(n^2)$ whether a graph G with $|V(G)| = n$ contains a triangle, then ENUM_p is not in $\mathsf{DelayC_{lin}}$ already for wdPTs in $\mathcal{C}_{\mathsf{pf}}$ consisting of only a root node with a single child where the root node contains a single binary atom, the child node contains two binary atoms, and the two nodes share two variables.*

Proof Idea. Constructing a pattern tree p with only two nodes R and N and three binary atoms and an appropriate database $\mathcal{D}$, we can decide whether a graph has a triangle by checking whether there is a $\mu \in p(\mathcal{D})$ such that μ is a mapping on the whole pattern tree. Using the fact that $|p|$ is constant, constant delay enumeration with linear preprocessing thus leads to an algorithm detecting a triangle in $\mathcal{O}(n^2)$. ◀

The negative result in Proposition 23 is mainly due to the restriction of preprocessing to linear time. Below, we show that if we relax this restriction and allow preprocessing in time $\mathcal{O}(|D|^m f(|p|))$ in terms of combined complexity for a constant $m > 0$ and some computable function f, then constant delay is achievable. Note that it is important to require that m be a constant in order to exclude preprocessing of time $\mathcal{O}(|\mathcal{D}|^{|p|})$, which in most cases would suffice to simply compute all of the solutions.

▶ **Theorem 24.** *Let $c, k \geq 1$ be positive integers. Then there exists an enumeration algorithm $\mathcal{A}$ for $\text{ENUM}(\ell\text{-}\mathsf{TW}(k) \cap \mathsf{BI}(c))$ with delay(0) in $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^{c+k+1})$ and delay(i) in $\mathcal{O}(1)$ for $i > 0$.*

Proof Idea. In the preprocessing, we construct a global tree decomposition of all atoms, which is consistent with the structure of the pattern tree (cf. our discussion preceding Theorem 5). Then by partitioning the corresponding relations of the nodes of the decomposition and eliminating tuples which are not part of some solution, a repeated top-down traversal through the tree yields all solutions with a delay only in the size of the pattern tree. ◀

Table 1 Summary of the main results on evaluation and enumeration of wdPTs.

	ℓ-TW(k) $\cap$ BI(c)	g-TW(k) $\cap$ SBI(c)	ℓ-TW(k) $\cap$ SBI(c)	g-TW(k)
p-Eval($\mathcal{C}$)	PTIME [4]	FPT	W[1]-complete	W[2]-hard
p-Max-Eval($\mathcal{C}$)	PTIME [4]	PTIME [4]	W[1]-hard	PTIME [4]
Enum($\mathcal{C}$)	SDelayP	*open*	not OutputP	not OutputP
Max-Enum($\mathcal{C}$)	not OutputP	not OutputP	not OutputP	not OutputP
p-Enum($\mathcal{C}$)	SDelayP	DelayFPT	not OutputFPT	not OutputFPT
p-Max-Enum($\mathcal{C}$)	DelayFPT	DelayFPT	not OutputFPT	DelayFPT

We note that the proof in fact allows to show that in the above theorem, $f(|p|)$ is actually polynomial w.r.t. $|p|$. Thus, given an exponential number of solutions, it is impossible to compute all of them during the preprocessing step.

For wdPTs with projection, we need to restrict the class of CQs in the pattern tree for a chance to achieve constant delay with linear preprocessing: Recall that – under reasonable complexity assumptions – the class of free-connex acyclic (respectively of bounded free-connex treewidth) CQs is the only class of acyclic (bounded treewidth) CQs which allows constant delay enumeration [3]. However, we show below that Enum_p is not in $\mathsf{DelayC_{lin}}$ even for wdPTs with such a restriction imposed locally on each set of atoms. As in Proposition 23, constant delay and linear preprocessing would lead to an unlikely upper bound on the runtime of a well-studied combinatorial problem.

▶ **Proposition 25.** *If the product AB of two Boolean $n \times n$ matrices A and B cannot be computed in time $\mathcal{O}(n^2)$, then Enum_p is not in $\mathsf{DelayC_{lin}}$ already for wdPTs consisting of only two nodes, each containing a single binary atom and sharing a single variable.*

Let $\mathcal{C}^*$ be the class of conjunctive queries with bounded free-connex treewidth. Then, even if we allow polynomial time preprocessing instead of a linear one as in Theorem 24, there is no constant delay enumeration algorithm for pattern trees in $\{g\text{-}\mathcal{C}^*, l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)\}$ assuming that $\mathsf{W[1]} \neq \mathsf{FPT}$. This is due to the fact that enumeration in these classes is not in $\mathsf{OutputFPT}$ under this complexity assumption.

▶ **Proposition 26.** *Let $c \geq 1$ and $\mathcal{C}^*$ be the class of conjunctive queries with bounded free-connex treewidth. Further let $p \in \{g\text{-}\mathcal{C}^*, l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)\}$ and $\mathcal{D}$ be a database. Then there is no positive integer m and computable function f such that $p(\mathcal{D})$ can be enumerated with delay(0) in $\mathcal{O}(|D|^m \cdot f(|p|))$ and delay(i) in $\mathcal{O}(1)$ for $i \geq 1$ unless $\mathsf{FPT} = \mathsf{W[1]}$.*

6 Conclusion

In this paper, we have embarked on a complexity analysis of two versions of the enumeration problem of wdPTs: the problems of enumerating all solutions and of enumerating the maximal solutions. Due to the close relationship with the parameterized complexity of the corresponding evaluation problem, we have also revisited the evaluation and max-evaluation problems. A summary of the main results is given in Table 1.

For the problems Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$) we have identified fixed-parameter tractable and intractable cases. Likewise, for the two variants of the enumeration problem, we have identified tractable and intractable cases – both in terms of classical and parameterized complexity. More precisely, for the classical complexity, we have established tractability

by showing a strong form of tractability (i.e., polynomial delay) and we have established intractability by ruling out even a weaker form of tractability (i.e., output polynomial time). We have proved analogous results from a parameterized complexity point of view.

Even though we have provided quite a comprehensive picture of the complexities in various settings, Table 1 still calls for further work. Above all, the ENUM($\mathcal{C}$) problem with $\mathcal{C} = g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$ is open. We conjecture tractability in this case – but this has to be proved yet. Also, for two of our W[1]- and W[2]-hardness results, a matching upper bound is missing. Finally, the search for tractable classes both, for evaluation and enumeration of wdPTs should be continued. Note that none of the restrictions studied here sufficed to ensure tractability of MAX-ENUM($\mathcal{C}$). Hence, further restrictions should be studied.

Acknowledgments. This work was supported by the Vienna Science and Technology Fund (WWTF) through project ICT12-015 and by the Austrian Science Fund (FWF):P25207-N23. Markus Kröll was funded by FWF project W1255-N23.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, Reading, Massachusetts, 1995.

2 Shqiponja Ahmetaj, Wolfgang Fischl, Reinhard Pichler, Mantas Simkus, and Sebastian Skritek. Towards reconciling SPARQL and certain answers. In *Proc. WWW 2015*, pages 23–33. ACM, 2015.

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

4 Pablo Barceló, Reinhard Pichler, and Sebastian Skritek. Efficient evaluation and approximation of well-designed pattern trees. In *Proc. PODS 2015*, pages 131–144. ACM, 2015.

5 Andrei A. Bulatov, Víctor Dalmau, Martin Grohe, and Dániel Marx. Enumerating homomorphisms. *J. Comput. Syst. Sci.*, 78(2):638–650, 2012.

6 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

7 Nadia Creignou and J-J Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique théorique et applications*, 31(6):499–511, 1997.

8 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *CoRR*, abs/1306.2171, 2013.

9 Nadia Creignou and Heribert Vollmer. Parameterized complexity of weighted satisfiability problems: Decision, enumeration, counting. *Fundam. Inform.*, 136(4):297–316, 2015. doi:10.3233/FI-2015-1159.

10 Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proc. CP*, pages 310–326, 2002.

11 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, Berlin/Heidelberg/New York, 1999.

12 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proc. PODS 2014*, pages 121–131. ACM, 2014.

13 Jörg Flum and Martin Grohe. Parameterized complexity theory. Berlin/Heidelberg/New York, 2010.

14 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

15 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

16 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), 2007.

17 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

18 Dimitris J Kavvadias, Martha Sideri, and Elias C Stavropoulos. Generating all maximal models of a boolean expression. *Information Processing Letters*, 74(3):157–162, 2000.

19 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proc. PODS 2013*, pages 297–308. ACM, 2013.

20 Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *Proc. ICDT 2015*, volume 31 of *LIPIcs*, pages 212–229. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

21 Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science*, 407(1):458–473, 2008.

22 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. ISAAC 2014*, pages 296–303. ACM, 2014.

23 Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.*, 38(4):25, 2013.

24 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

25 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

26 Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed SPARQL. In *Proc. PODS 2014*, pages 39–50. ACM, 2014.

27 Reinhard Pichler and Sebastian Skritek. On the hardness of counting the solutions of SPARQL queries. In *Proc. AMW 2014*, volume 1189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

28 Eric Prud′hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, World Wide Web Consortium (W3C), January 2008. URL: http://www.w3.org/TR/rdf-sparql-query/.

29 Luc Segoufin. Enumerating with constant delay the answers to a query. In *Proc. ICDT 2013*, pages 10–20. ACM, 2013.

30 Luc Segoufin. A glimpse on constant delay enumeration (invited talk). In *Proc. STACS 2014*, volume 25 of *LIPIcs*, pages 13–27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.

31 Yann Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Universite Paris Diderot – Paris 7, December 2010. URL: http://www.prism.uvsq.fr/~ystr/these_strozecki.

32 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB*, pages 82–94, 1981.

A Practically Efficient Algorithm for Generating Answers to Keyword Search Over Data Graphs*†

Konstantin Golenberg[1] and Yehoshua Sagiv[2]

1 **The Hebrew University of Jerusalem, Jerusalem, Israel**
2 **The Hebrew University of Jerusalem, Jerusalem, Israel**

Abstract

In keyword search over a data graph, an answer is a non-redundant subtree that contains all the keywords of the query. A naive approach to producing all the answers by increasing height is to generalize Dijkstra's algorithm to enumerating all acyclic paths by increasing weight. The idea of *freezing* is introduced so that (most) non-shortest paths are generated only if they are actually needed for producing answers. The resulting algorithm for generating subtrees, called *GTF*, is subtle and its proof of correctness is intricate. Extensive experiments show that GTF outperforms existing systems, even ones that for efficiency's sake are incomplete (i.e., cannot produce all the answers). In particular, GTF is scalable and performs well even on large data graphs and when many answers are needed.

1998 ACM Subject Classification H.3.3 [Information Storage and Retrieval] Information Search and Retrieval – Search process, H.2.4 [Database Management] Systems – Query processing

Keywords and phrases Keyword search over data graphs, subtree enumeration by height, top-k answers, efficiency

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.23

1 Introduction

Keyword search over data graphs is a convenient paradigm of querying semistructured and linked data. Answers, however, are similar to those obtained from a database system, in the sense that they are succinct (rather than just relevant documents) and include semantics (in the form of entities and relationships) and not merely free text. Data graphs can be built from a variety of formats, such as XML, relational databases, RDF and social networks. They can also be obtained from the amalgamation of many heterogeneous sources. When it comes to querying data graphs, keyword search alleviates their lack of coherence and facilitates easy search for precise answers, as if users deal with a traditional database system.

In this paper, we address the issue of efficiency. Computing keyword queries over data graphs is much more involved than evaluation of relational expressions. Quite a few systems have been developed (see [2] for details). However, they fall short of the degree of efficiency and scalability that is required in practice. Some algorithms sacrifice *completeness* for the sake of efficiency; that is, they are not capable of generating all the answers and, consequently, may miss some relevant ones.

We present a novel algorithm, called *Generating Trees with Freezing* (GTF). We start with a straightforward generalization of Dijkstra's shortest-path algorithm to the task of

* This work was supported by the Israel Science Foundation (Grant No. 1632/12).
† The full version of this paper appears in [5].

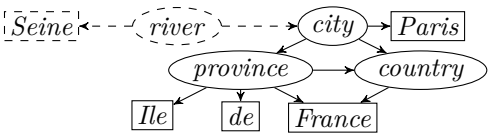


Figure 1 A snippet of a data graph.

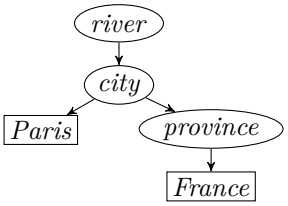


Figure 2 Redundant subtree.

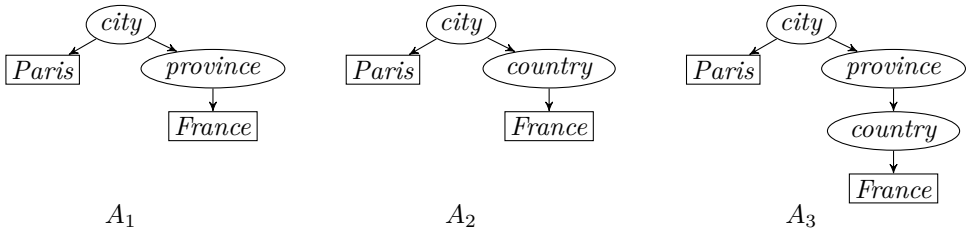


Figure 3 Answers.

constructing all simple (i.e., acyclic) paths, rather than just the shortest ones. Our main contribution is incorporating the *freezing* technique that enhances efficiency by up to one order of magnitude, compared with the naive generalization of Dijkstra's algorithm. The main idea is to avoid the construction of most non-shortest paths until they are actually needed in answers. Freezing may seem intuitively clear, but making it work involves subtle details and requires an intricate proof of correctness.

Our main theoretical contribution is the algorithm GTF, which incorporates freezing, and its proof of correctness. Our main practical contribution is showing experimentally (in Section 5 and Appendix B of [5]) that GTF is both more efficient and more scalable than existing systems. This contribution is especially significant in light of the following. First, GTF is complete (i.e., it does not miss answers); moreover, we show experimentally that not missing answers is important in practice. Second, the order of generating answers is by increasing height. This order is commonly deemed a good strategy for an initial ranking that is likely to be in a good correlation with the final one (i.e., by increasing weight).

2 Preliminaries

We model data as a directed graph G, similarly to [1]. Data graphs can be constructed from a variety of formants (e.g., RDB, XML and RDF). Nodes represent entities and relationships, while edges correspond to connections among them (e.g., foreign-key references when the data graph is constructed from a relational database). We assume that text appears only in the nodes. This is not a limitation, because we can always split an edge (with text) so that it passes through a node. Some nodes are for keywords, rather than entities and relationships. In particular, for each keyword k that appears in the data graph, there is a dedicated node. By a slight abuse of notation, we do not distinguish between a keyword k and its node – both are called *keyword* and denoted by k. For all nodes v of the data graph that contain a keyword k, there is a directed edge from v to k. Thus, keywords have only incoming edges.

Figure 1 shows a snippet of a data graph. The dashed part should be ignored unless explicitly stated otherwise. Ordinary nodes are shown as ovals. For clarity, the type of each node appears inside the oval. Keyword nodes are depicted as rectangles. To keep the figure

small, only a few of the keywords that appear in the graph are shown as nodes. For example, a type is also a keyword and has its own node in the full graph. For each oval, there is an edge to every keyword that it contains.

Let $G = (V, E)$ be a directed data graph, where V and E are the sets of nodes and edges, respectively. A directed path is denoted by $\langle v_1, \ldots, v_m \rangle$. We only consider *rooted* (and, hence, directed) subtrees T of G. That is, T has a unique node r, such that for all nodes u of T, there is exactly one path in T from r to u. Consider a query K, that is, a set of at least two keywords. A *K-subtree* is a rooted subtree of G, such that its leaves are exactly the keywords of K. We say that a node $v \in V$ is a *K-root* if it is the root of some K-subtree of G. It is observed in [1] that v is a K-root if and only if for all $k \in K$, there is a path in G from v to k. An *answer* to K is a K-subtree T that is *non-redundant* (or *reduced*) in the sense that no proper subtree T' of T is also a K-subtree. It is easy to show that a K-subtree T of G is an answer if and only if the root of T has at least two children. Even if v is a K-root, it does not necessarily follow that there is an answer to K that is rooted at v (because it is possible that in all K-subtrees rooted at v, there is only one child of v).

Figure 3 shows three answers to the query $\{France, Paris\}$ over the data graph of Figure 1. The answer A_1 means that the city Paris is located in a province containing the word France in its name. The answer A_2 states that the city Paris is located in the country France. Finally, the answer A_3 means that Paris is located in a province which is located in France.

Now, consider also the dashed part of Figure 1, that is, the keyword *Seine* and the node *river* with its outgoing edges. There is a path from *river* to every keyword of $K = \{France, Paris\}$. Hence, *river* is a K-root. However, the K-subtree of Figure 2 is not an answer to K, because its root has only one child.

For ranking, the nodes and edges of the data graph have positive weights. The *weight* of a path (or a tree) is the sum of weights of all its nodes and edges. The rank of an answer is inversely proportional to its weight. The *height* of a tree is the maximal weight over all paths from the root to any leaf (which is a keyword of the query). For example, suppose that the weight of each node and edge is 1. The heights of the answers A_1 and A_3 (of Figure 3) are 5 and 7, respectively. In A_1, the path from the root to *France* is a minimal (i.e., shortest) one between these two nodes, in the whole graph, and its weight is 5. In A_3, however, the path from the root (which is the same as in A_1) to *France* has a higher weight, namely, 7.

3 The GTF Algorithm

3.1 The Naive Approach

Consider a query $K = \{k_1, \ldots, k_n\}$. In [1], they use a backward shortest-path iterator from each keyword node k_i. That is, starting at each k_i, they apply Dijkstra's shortest-path algorithm in the opposite direction of the edges. If a node v is reached by the backward iterators from all the k_i, then v is a K-root (and, hence, might be the root of some answers). In this way, answers are generated by increasing height. However, this approach can only find answers that consist of shortest paths from the root to the keyword nodes. Hence, it misses answers (e.g., it cannot produce A_3 of Figure 3).

Dijkstra's algorithm can be straightforwardly generalized to construct all the simple (i.e., acyclic) paths by increasing weight. This approach is used[1] in [11] and it consists of two parts: path construction and answer production. Each constructed path is from

[1] They used it on a small *summary* graph to construct database queries from keywords.

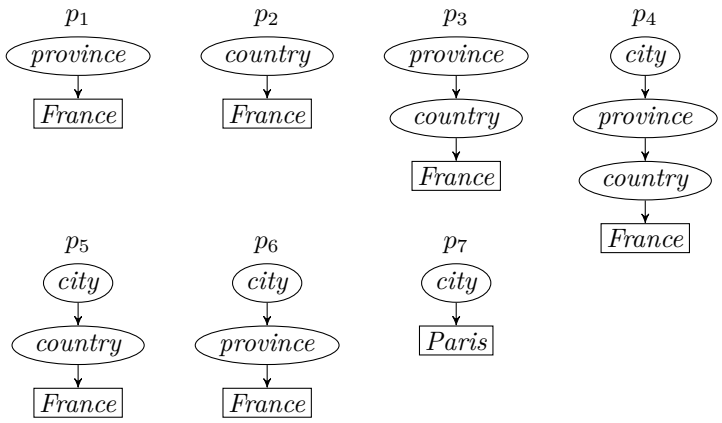


Figure 4 Paths to keywords in the graph snippet of Figure 1.

some node of G to a keyword of K. Since paths are constructed backwards, the algorithm starts simultaneously from all the keyword nodes of K. It uses a single priority queue to generate, by increasing weight, all simple paths to every keyword node of K. When the algorithm discovers that a node v is a K-root (i.e., there is a path from v to every k_i), it starts producing answers rooted at v. This is done by considering every combination of paths $p_1, \ldots, p_n$, such that p_i is from v to k_i $(1 \leq i \leq n)$. If the combination is a non-redundant K-subtree of G, then it is produced as an answer. It should be noted that in [11], answers are subgraphs; hence, every combination of paths $p_1, \ldots, p_n$ is an answer. We choose to produce subtrees as answers for two reasons. First, in the experiments of Section 5, we compare our approach with other systems that produce subtrees. Second, it is easier for users to understand answers that are presented as subtrees, rather than subgraphs.

The drawback of the above approach is constructing a large number of paths that are never used in any of the generated answers. To overcome this problem, the next section introduces the technique of *freezing*, thereby most non-minimal paths are generated only if they are actually needed to produce answers. Section 3.3 describes the algorithm *Generating Trees with Freezing* (GTF) that employs this technique.

To save space (when constructing all simple paths), we use the common technique known as *tree of paths*. In particular, a path p is a linked list, such that its first node points to the rest of p. As an example, consider the graph snippet of Figure 1. The paths that lead to the keyword *France* are p_1, p_2, p_3, p_4, p_5 and p_6, shown in Figure 4. Their tree of paths is presented in Figure 5.

Since we build paths backwards, a data graph is preprocessed to produce for each node v the set of its *parents*, that is, the set of nodes v', such that (v', v) is an edge of the data graph. We use the following notation. Given a path p that starts at a node v, the extension of p with a parent v' of v is denoted by $v' \rightarrow p$. Note that v' is the first node of $v' \rightarrow p$ and v is the second one.

3.2 Incorporating Freezing

The general idea of freezing is to avoid the construction of paths that cannot contribute to production of answers. To achieve that, a non-minimal path p is frozen until it is certain that p can reach (when constructed backwards) a K-root. In particular, the first path that reaches a node v is always a minimal one. When additional paths reach v, they are frozen

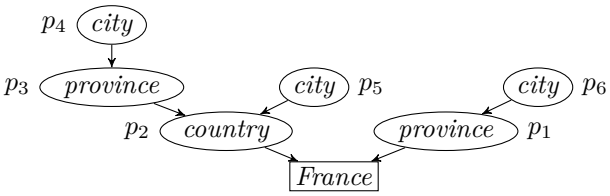


Figure 5 Tree of paths.

there until v is discovered to be on a path from a K-root to a keyword node. The process of answer production in the GTF algorithm remains the same as in the naive approach.

We now describe some details about the implementation of GTF. We mark nodes of the data graph as either *active*, *visited* or *in-answer*. Since we simultaneously construct paths to all the keywords (of the query $K = \{k_1, \ldots, k_n\}$), a node has a separate mark for each keyword. The marks of a node v are stored in the array $v.marks$, which has an entry for each keyword. For a keyword k_i, the mark of v (i.e., $v.marks[k_i]$) means the following. Node v is *active* if we have not yet discovered that there is a path from v to k_i. Node v is *visited* if a minimal path from v to k_i has been produced. And v is marked as *in-answer* when we discover for the first time that v is on a path from some K-root to k_i.

If $v.marks[k_i]$ is *visited* and a path p from v to k_i is removed from the queue, then p is *frozen* at v. Frozen paths from v to k_i are stored in a dedicated list $v.frozen[k_i]$. The paths of $v.frozen[k_i]$ are *unfrozen* (i.e., are moved back into the queue) when $v.marks[k_i]$ is changed to *in-answer*.

We now describe the execution of GTF on the graph snippet of Figure 1, assuming that the query is $K = \{France, Paris\}$. Initially, two paths $\langle France\rangle$ and $\langle Paris\rangle$, each consisting of one keyword of K, are inserted into the queue, where lower weight means higher priority. Next, the top of the queue is removed; suppose that it is $\langle France\rangle$. First, we change $France.marks[France]$ to *visited*. Second, for each parent v of *France*, the path $v \to France$ is inserted into the queue; namely, these are the paths p_1 and p_2 of Figure 4. We continue to iterate in this way. Suppose that now $\langle Paris\rangle$ has the lowest weight. So, it is removed from the queue, $Paris.marks[Paris]$ is changed to *visited*, and the path p_7 (of Figure 4) is inserted into the queue.

Now, let the path p_1 be removed from the queue. As a result, $province.marks[France]$ is changed to *visited*, and the path $p_6 = city \to p_1$ is inserted into the queue. Next, assume that p_2 is removed from the queue. So, $country.marks[France]$ is changed to *visited*, and the paths $p_3 = province \to p_2$ and $p_5 = city \to p_2$ are inserted into the queue.

Now, suppose that p_3 is at the top of the queue. So, p_3 is removed and immediately frozen at *province* (i.e., added to $province.frozen[France]$), because $province.marks[France] = visited$. Consequently, no paths are added to the queue in this iteration. Next, assume that p_6 is removed from the queue. The value of $city.marks[France]$ is changed to *visited* and no paths are inserted into the queue, because *city* has no incoming edges.

Now, suppose that p_7 is at the top of the queue. So, it is removed and $city.marks[Paris]$ is changed to *visited*. Currently, both $city.marks[Paris]$ and $city.marks[France]$ are *visited*. That is, there is a path from *city* to all the keywords of the query $\{France, Paris\}$. Recall that the paths that have reached *city* so far are p_6 and p_7. For each one of those paths p, the following is done, assuming that p ends at the keyword k. For each node v of p, we change the mark of v for k to *in-answer* and unfreeze paths to k that are frozen at v. Doing it for p_6 means that $city.marks[France]$, $province.marks[France]$ and $France.marks[France]$ are all changed to *in-answer*. In addition, the path p_3 is removed from $province.frozen[France]$

and inserted back into the queue. We act similarly on p_7. That is, *city.marks*[*Paris*] and *Paris.marks*[*Paris*] are changed to *in-answer*. In this case, there are no paths to be unfrozen.

Now, the marks of *city* for all the keywords (of the query) are *in-answer*. Hence, we generate answers from the paths that have already reached *city*. As a result, the answer A_1 of Figure 3 is produced. Moreover, from now on, when a new path reaches *city*, we will try to generate more answers by applying **produceAnswers**$(\mathcal{P}, p)$.

3.3 The Pseudocode of the GTF Algorithm

The GTF algorithm is presented in Figure 6 and its helper procedures in Figure 7. The input is a data graph $G = (V, E)$ and a query $K = \{k_1, \ldots, k_n\}$. The algorithm uses a single priority queue Q to generate, by increasing weight, all simple paths to every keyword node of K. For each node $v \in V$, there is a flag *isKRoot* that indicates whether v has a path to each keyword of K. Initially, that flag is **false**. For each node $v \in V$, the set of the constructed paths from v to the keyword k is stored in $v.paths[k]$, which is initially empty. Also, for all the keywords of K and nodes of G, we initialize the marks to be *active* and the lists of frozen paths to be empty. The paths are constructed backwards, that is, from the last node (which is always a keyword). Therefore, for each $k \in K$, we insert the path $\langle k \rangle$ (consisting of the single node k) into Q. All these initializations are done in lines 1–9 (of Figure 6).

The main loop of lines 10–37 is repeated while Q is not empty. Line 11 removes the best (i.e., least-weight) path p from Q. Let v and k_i be the first and last, respectively, nodes of p. Line 12 freezes p provided that it has to be done. This is accomplished by calling the procedure **freeze**(p) of Figure 7 that operates as follows. If the mark of v for k_i is *visited*, then p is frozen at v by adding it to $v.frozen[k_i]$ and **true** is returned; in addition, the main loop continues (in line 13) to the next iteration. Otherwise, **false** is returned and p is handled as we describe next.

Line 15 checks if p is the first path from v to k_i that has been removed from Q. If so, line 16 changes the mark of v for k_i from *active* to *visited*. Line 17 assigns **true** to the flag *relax*, which means that (as of now) p should spawn new paths that will be added to Q.

The test of line 18 splits the execution of the algorithm into two cases. If v is a K-root (which must have been discovered in a previous iteration and means that for every $k \in K$, there is a path from v to k), then the following is done. First, line 19 calls the procedure **unfreeze**(p, Q) of Figure 7 that unfreezes (i.e., inserts into Q) all the paths to k_i that are frozen at nodes of p (i.e., the paths of $\bar{v}.frozen[k_i]$, where $\bar{v}$ is a node of p). In addition, for all nodes $\bar{v}$ of p, the procedure **unfreeze**(p, Q) changes the mark of $\bar{v}$ for k_i to *in-answer*. Second, line 20 tests whether p is acyclic. If so, line 21 adds p to the paths of v that reach k_i, and line 22 produces new answers that include p by calling **produceAnswers** of Figure 7. The pseudocode of **produceAnswers**$(v.paths, p)$ is just an efficient implementation of considering every combination of paths $p_1, \ldots, p_n$, such that p_i is from v to k_i $(1 \leq i \leq n)$, and checking that it is an answer to K. (It should be noted that GTF generates answers by increasing height.) If the test of line 20 is **false**, then the flag *relax* is changed back to **false**, thereby ending the current iteration of the main loop.

If the test of line 18 is **false** (i.e., v has not yet been discovered to be a K-root), the execution continues in line 26 that adds p to the paths of v that reach k_i. Line 27 tests whether v is now a K-root and if so, the flag *isKRoot* is set to **true** and the following is done. The nested loops of lines 29–32 iterate over all paths p' (that have already been discovered) from v to any keyword node of K (i.e., not just k_i). For each p', where k' is the last node of p' (and, hence, is a keyword), line 31 calls **unfreeze**(p', Q), thereby inserting into Q all the paths to k' that are frozen at nodes of p' and changing the mark (for k') of those nodes to

Algorithm: *GTF (Generate Trees with Freezing)*
Input: $G = (V, E)$ is a data graph
K is a set of keyword nodes
Output: Answers to K

```
1: Q ← an empty priority queue
2: for v ∈ V do
3:    v.isKRoot ← false
4: for v ∈ V and k ∈ K do
5:    v.paths[k] ← ∅
6:    v.frozen[k] ← ∅
7:    v.marks[k] ← active
8: for k ∈ K do
9:    Q.insert(⟨k⟩)
10: while Q is not empty do
11:    p ← Q.remove()
12:    if freeze(p) then
13:       continue
14:    v ← first(p)
15:    if v.marks[p.keyword] = active then
16:       v.marks[p.keyword] ← visited
17:    relax ← true
18:    if v.isKRoot = true then
19:       unfreeze(p, Q)
20:       if p has no cycles then
21:          v.paths[p.keyword].add(p)
22:          produceAnswers(v.paths, p)
23:       else
24:          relax ← false
25:    else
26:       v.paths[p.keyword].add(p)
27:       if for all k ∈ K, it holds that v.paths[k] ≠ ∅ then
28:          v.isKRoot ← true
29:          for k ∈ K do
30:             for p′ ∈ v.pahts[k] do
31:                unfreeze(p′, Q)
32:             remove cyclic paths from v.paths[k]
33:          produceAnswers(v.paths, p)
34:    if relax then
35:       for v′ ∈ parents(v) do
36:          if v′ is not on p or v′ → p is essential then
37:             Q.insert(v′ → p)
```

Figure 6 The GTF algorithm.

Procedure: **freeze**(p)

```
1: if first(p).marks[p.keyword] = visited then
2:     first(p).frozen[p.keyword].add(p)
3:     return true
4: else
5:     return false
```

Procedure: **unfreeze**(p, Q)

```
1: p' ← p
2: while p' ≠ ⊥ do
3:     v̄ ← first(p')
4:     if v̄.marks[p.keyword] ≠ in-answer then
5:         v̄.marks[p.keyword] ← in-answer
6:         for p'' ∈ v̄.frozen[p.keyword] do
7:             Q.insert(p'')
8:     p' ← predecessor(p')
```

Procedure: **produceAnswers**$(\mathcal{P}, p)$
Output: answers rooted at **first**(p)

```
1: P[p.keyword] ← {p}
2: iter ← new pathGroups(P)
3: while iter.hasNext() do
4:     P̄ ← iter.next()
5:     a ← combine all the paths in P̄
       /* next() ensures that the combination of all the paths in P̄ yields a
       tree (rather than a graph) */
6:     if the root of a has more than one
       child then
7:         output a
```

Figure 7 Helper procedures for the GTF algorithm.

in-answer. Line 32 removes all the cyclic paths among those stored at v. Line 33 generates answers from the paths that remain at v.

If the test of line 34 is **true**, the relaxation of p is done in lines 35–37 as follows. For each parent v' of v, the path $v' \to p$ is inserted into Q if either one of the following two holds (as tested in line 36). First, v' is not on p. Second, $v' \to p$ is essential, according to the following definition. The path $v' \to p$ is *essential* if v' appears on p and the section of $v' \to p$ from its first node (which is v') to the next occurrence of v' has at least one node u, such that $u.marks[k] = visited$, where the keyword k is the last node of p. Appendix A of [5] gives an example that shows why essential paths (which are cyclic) have to be inserted into Q.

Note that due to line 24, no cyclic path $p[v, k]$ is relaxed if v has already been discovered to be a K-root in a previous iteration. The reason is that none of the nodes along $p[v, k]$ could have the mark *visited* for the keyword k (hence, no paths are frozen at those nodes).

Observe that before v is known to be a K-root, we add cyclic paths to the array $v.paths$. Only when discovering that v is a K-root, do we remove all cyclic paths from $v.paths$ (in line 32) and stop adding them in subsequent iterations. This is lazy evaluation, because prior to knowing that answers with the K-root v should be produced, it is a waste of time to test whether paths from v are cyclic.

4 Correctness and Complexity of GTF

4.1 Definitions and Observations

Before proving correctness of the GTF algorithm, we define some notation and terminology (in addition to those of Section 2) and state a few observations. Recall that the data graph is $G = (V, E)$. Usually, a keyword is denoted by k, whereas r, u, v and z are any nodes of V.

We only consider directed paths of G that are defined as usual. If p is a path from v to k, then we write it as $p[v, k]$ when we want to explicitly state its first and last nodes. We say that node u is *reachable* from v if there is a path from v to u.

A *suffix* of $p[v, k]$ is a traversal of $p[v, k]$ that starts at (some particular occurrence of) a node u and ends at the last node of p. Hence, a suffix of $p[v, k]$ is denoted by $p[u, k]$. A *prefix*

of $p[v,k]$ is a traversal of $p[v,k]$ that starts at v and ends at (some particular occurrence of) a node u. Hence, a prefix of $p[v,k]$ is denoted by $p[v,u]$. A suffix or prefix of $p[v,k]$ is *proper* if it is different from $p[v,k]$ itself.

Consider two paths $p_1[v,z]$ and $p_2[z,u]$; that is, the former ends in the node where the latter starts. Their *concatenation*, denoted by $p_1[v,z] \circ p_2[z,u]$, is obtained by joining them at node z.

As already mentioned in Section 2, a positive *weight* function w is defined on the nodes and edges of G. The weight of a path $p[v,u]$, denoted by $w(p[v,u])$, is the sum of weights over all the nodes and edges of $p[v,u]$. A *minimal* path from v to u has the minimum weight among all paths from v to u. Since the weight function is positive, there are no zero-weight cycles. Therefore, a minimal path is acyclic. Also observe that the weight of a proper suffix or prefix is strictly smaller than that of the whole path.[2]

Let K be a query (i.e., a set of at least two keywords). Recall from Section 2 the definitions of K-root, K-subtree and height of a subtree. The *best height* of a K-root r is the maximum weight among all the minimal paths from r to any keyword $k \in K$. Note that the height of any K-subtree rooted at r is at least the best height of r.

Consider a nonempty set of nodes S and a node v. If v is reachable from every node of S, then we say that node $u \in S$ is *closest to* v if a minimal path from u to v has the minimum weight among all paths from any node of S to v.

Similarly, if every node of S is reachable from v, then we say that node $u \in S$ is *closest from* v if a minimal path from v to u has the minimum weight among all paths from v to any node of S.

In the sequel, line numbers refer to the algorithm GTF of Figure 6, unless explicitly stated otherwise. We say that a node $v \in V$ is *discovered as a K-root* if the test of line 27 is satisfied and $v.isRoot$ is assigned **true** in line 28. Observe that the test of line 27 is **true** if and only if for all $k \in K$, it holds that $v.marks[k]$ is either *visited* or *in-answer*. Also note that line 28 is executed at most once for each node v of G. Thus, there is at most one iteration of the main loop (i.e., line 10) that discovers v as K-root.

We say that a path p is *constructed* when it is inserted into Q for the first time, which must happen in line 37. A path is *exposed* when it is removed from Q in line 11. Observe that a path $p[v,k]$ may be exposed more than once, due to freezing and unfreezing.

▶ **Proposition 1.** *A path can be exposed at most twice.*

Proof. When an iteration exposes a path $p[v,k]$ for the first time, it does exactly one of the following. It freezes $p[v,k]$ at node v, discard $p[v,k]$ due to line 24, or extend (i.e., relax) $p[v,k]$ in the loop of line 35 and inserts the results into Q in line 37. Note that some relaxations of $p[v,k]$ are never inserted into Q, due to the test of line 36. Only if $p[v,k]$ is frozen at v, can it be inserted a second time into Q, in line 7 of the procedure **unfreeze** (Figure 7) that also sets $v.marks[k]$ to *in-answer*. But then $p[v,k]$ cannot freeze again at v, because $v.marks[k]$ does not change after becoming *in-answer*. Therefore, $p[v,k]$ cannot be inserted into Q a third time. ◀

In the next section, we sometimes refer to the mark of a node v of a path p. It should be clear from the context that we mean the mark of v for the keyword where p ends.

[2] For the proof of correctness, it is enough for the weight function to be non-negative (rather than positive) provided that every cycle has a positive weight.

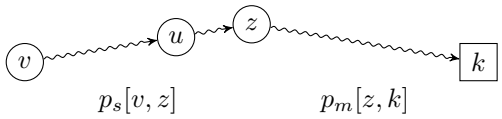


Figure 8 The path $\bar{p}[v, k]$.

4.2 The Proof

We start with an auxiliary lemma that considers the concatenation of two paths, where the linking node is z, as shown in Figure 8 (note that a wavy arrow denotes a path, rather than a single edge). Such a concatenation is used in the proofs of subsequent lemmas.

▶ **Lemma 2.** *Let k be a keyword of the query K, and let v and z be nodes of the data graph. Consider two paths $p_s[v, z]$ and $p_m[z, k]$. Let $\bar{p}[v, k]$ be their concatenation at node z, that is,*

$$\bar{p}[v, k] = p_s[v, z] \circ p_m[z, k].$$

Suppose that the following hold at the beginning of iteration i of the main loop (line 10).

1. *The path $p_s[v, z]$ is minimal or (at least) acyclic.*
2. *The path $p_m[z, k]$ has changed $z.marks[k]$ from active to visited in an earlier iteration.*
3. *$z.marks[k] =$ visited.*
4. *For all nodes $u \neq z$ on the path $p_s[v, z]$, the suffix $\bar{p}[u, k]$ is not frozen at u.*
5. *The path $\bar{p}[v, k]$ has not yet been exposed.*

Then, some suffix of $\bar{p}[v, k]$ must be on Q at the beginning of iteration i.

Proof. Suppose, by way of contradiction, that no suffix of $\bar{p}[v, k]$ is on Q at the beginning of iteration i. Since $\bar{p}[v, k]$ has not yet been exposed, there are two possible cases regarding its state. We derive a contradiction by showing that none of them can happen.

Case 1: Some suffix of $\bar{p}[v, k]$ is frozen. This cannot happen at any node of $\bar{p}[z, k]$ (which is the same as $p_m[z, k]$), because Condition 3 implies that $p_m[z, k]$ has already changed $z.marks[k]$ to *visited*. Condition 4 implies that it cannot happen at the other nodes of $\bar{p}[v, k]$ (i.e., the nodes u of $p_s[v, z]$ that are different from z).

Case 2: Some suffix of $\bar{p}[v, k]$ has already been discarded (in an earlier iteration) either by the test of line 36 or due to line 24. This cannot happen to any suffix of $\bar{p}[z, k]$ (which is the same as $p_m[z, k]$), because $p_m[z, k]$ has already changed $z.marks[k]$ to *visited*. We now show that it cannot happen to any other suffix $\bar{p}[u, k]$, where u is a node of $p_s[v, z]$ other than z. Note that $\bar{p}[v, k]$ (and hence $\bar{p}[u, k]$) is not necessarily acyclic. However, the lemma states that $p_s[v, z]$ is acyclic. Therefore, if the suffix $\bar{p}[u, k]$, has a cycle that includes u, then it must also include z. But $z.marks[k]$ is *visited* from the moment it was changed to that value until the beginning of iteration i (because a mark cannot be changed to *visited* more than once). Hence, the suffix $\bar{p}[u, k]$ could not have been discarded by the test of line 36. It is also not possible that line 24 has already discarded $\bar{p}[u, k]$ for the following reason. If line 24 is reached (in an iteration that removed $\bar{p}[u, k]$ from Q), then for all nodes x on $\bar{p}[u, k]$, line 19 has already changed $x.marks[k]$ to *in-answer*. Therefore, $z.marks[k]$ cannot be *visited* at the beginning of iteration i.

It thus follows that some suffix of $\bar{p}[v, k]$ is on Q at the beginning of iteration i. ◀

▶ **Lemma 3.** *For all nodes $v \in V$ and keywords $k \in K$, the mark $v.marks[k]$ can be changed from active to visited only by a minimal path from v to k.*

Proof. Suppose that the lemma is not true for some keyword $k \in K$. Let v be a closest node to k among all those violating the lemma with respect to k. Node v is different from k, because the path $\langle k \rangle$ marks k as *visited*. We will derive a contradiction by showing that a minimal path changes $v.marks[k]$ from *active* to *visited*.

Let $p_s[v, k]$ be a minimal path from v to k. Consider the iteration i of the main loop (line 10 in Figure 6) that changes $v.marks[k]$ to *visited* (in line 16). Among all the nodes of $p_s[v, k]$ in which suffixes of some minimal paths from v to k are frozen at the beginning of iteration i, let z be the first one when traversing $p_s[v, k]$ from v to k (i.e., on the path $p_s[v, z]$, node z is the only one in which such a suffix is frozen). Node z exists for the following three reasons.

- The path $p_s[v, k]$ has not been exposed prior to iteration i, because we assume that $v.marks[k]$ is changed to *visited* in iteration i and that change can happen only once.
- The path $p_s[v, k]$ is acyclic (because it is minimal), so a suffix of $p_s[v, k]$ could not have been discarded either by the test of line 36 or due to line 24.
- The path $p_s[v, k]$ (or any suffix thereof) cannot be on the queue at the beginning of iteration i, because v violates the lemma, which means that a non-minimal path from v to k must be removed from the queue at the beginning of that iteration.

The above three observations imply that a proper suffix of $p_s[v, k]$ must be frozen at the beginning of iteration i and, hence, node z exists. Observe that z is different from v, because a path to k can be frozen only at a node $\hat{v}$, such that $\hat{v}.marks[k] = visited$, whereas we assume that $v.marks[k]$ is *active* at the beginning of iteration i.

By the selection of v and $p_s[v, k]$ (and the above fact that $z \neq v$), node z does not violate the lemma, because $p_s[z, k]$ is a proper suffix of $p_s[v, k]$ and, hence, z is closer to k than v. Therefore, according to the lemma, there is a minimal path $p_m[z, k]$ that changes $z.marks[k]$ to *visited*. Consequently,

$$w(p_m[z, k]) \leq w(p_s[z, k]). \tag{1}$$

Now, consider the path

$$\bar{p}[v, k] = p_s[v, z] \circ p_m[z, k]. \tag{2}$$

Since $p_s[v, k]$ is a minimal path from v to k, Equations (1) and (2) imply that so is $\bar{p}[v, k]$.

We now show that the conditions of Lemma 2 are satisfied at the beginning of iteration i. In particular, Condition 1 holds, because $p_s[v, k]$ is acyclic (since it is minimal) and, hence, so is the path $p_s[v, z]$. Condition 2 is satisfied, because of how $p_m[z, k]$ is defined. Condition 3 holds, because we chose z to be a node where a path to k is frozen. Condition 4 is satisfied, because of how z was chosen and the fact that $\bar{p}[v, k]$ is minimal. Condition 5 is satisfied, because we have assumed that $v.marks[k]$ is changed from *active* to *visited* during iteration i.

By Lemma 2, a suffix of $\bar{p}[v, k]$ must be on the queue at the beginning of iteration i. This contradicts our assumption that a non-minimal path (which has a strictly higher weight than any suffix of $\bar{p}[v, k]$) changes $v.marks[k]$ from *active* to *visited* in iteration i. ◀

▶ **Lemma 4.** *For all nodes $v \in V$ and keywords $k \in K$, such that k is reachable from v, if $v.marks[k]$ is active at the beginning of an iteration of the main loop (line 10), then Q contains a suffix (which is not necessarily proper) of a minimal path from v to k.*

Proof. The lemma is certainly true at the beginning of the first iteration, because the path $\langle k \rangle$ is on Q. Suppose that the lemma does not hold at the beginning of iteration i. Thus, every minimal path $p[v, k]$ has a proper suffix that is frozen at the beginning of iteration i.

(Note that a suffix of a minimal path cannot be discarded either by the test of line 36 or due to line 24, because it is acyclic.) Let z be the closest node from v having such a frozen suffix. Hence, $z.marks[k]$ is *visited* and $z \neq v$ (because $v.marks[k]$ is *active*). By Lemma 3, a minimal path $p_m[z,k]$ has changed $z.marks[k]$ to *visited*. Let $p_s[v,z]$ be a minimal path from v to z. Consider the path

$$\bar{p}[v,k] = p_s[v,z] \circ p_m[z,k].$$

The weight of $\bar{p}[v,k]$ is no more than that of a minimal path from v to k, because both $p_s[v,z]$ and $p_m[v,k]$ are minimal and the choice of z implies that it is on some minimal path from v to k. Hence, $\bar{p}[v,k]$ is a minimal path from v to k.

We now show that the conditions of Lemma 2 are satisfied. Conditions 1–3 clearly hold. Condition 4 is satisfied because of how z is chosen and the fact that $\bar{p}[v,k]$ is minimal. Condition 5 holds because $v.marks[k]$ is *active* at the beginning of iteration i.

By Lemma 2, a suffix of $\bar{p}[v,k]$ is on Q at the beginning of iteration i, contradicting our initial assumption. ◀

▶ **Lemma 5.** *Any constructed path can have at most $2n(n+1)$ nodes, where $n = |V|$ (i.e., the number of nodes in the graph). Hence, the algorithm constructs at most $(n+1)^{2n(n+1)}$ paths.*

Proof. We say that $v_m \to \cdots \to v_1$ is a *repeated run* in a path $\bar{p}$ if some suffix (not necessarily proper) of $\bar{p}$ has the form $v_m \to \cdots \to v_1 \to p$, where each v_i also appears in any two positions of p. In other words, for all i $(1 \leq i \leq m)$, the occurrence of v_i in $v_m \to \cdots \to v_1$ is (at least) the third one in the suffix $v_m \to \cdots \to v_1 \to p$. (We say that it is the third, rather than the first, because paths are constructed backwards).

When a path $p'[v',k']$ reaches a node v' for the third time, the mark of v' for the keyword k' has already been changed to *in-answer* in a previous iteration. This follows from the following two observations. First, the first path to reach a node v' is also the one to change its mark to *visited*. Second, a path that reaches a node marked as *visited* can be unfrozen only when that mark is changed to *in-answer*.

Let $v_m \to \cdots \to v_1$ be a repeated run in $\bar{p}$ and suppose that $m > n = |V|$. Hence, there is a node v_i that appears twice in the repeated run; that is, there is a $j < i$, such that $v_j = v_i$. If the path $v_i \to \cdots \to v_1 \to p$ is considered in the loop of line 35, then it would fail the test of line 36 (because, as explained earlier, all the nodes on the cycle $v_i \to \cdots \to v_j$ are already marked as *in-answer*). We conclude that the algorithm does not construct paths that have a repeated run with more than n nodes.

It thus follows that two disjoint repeated runs of a constructed path $\bar{p}$ must be separated by a node that appears (in a position between them) for the first or second time. A path can have at most $2n$ positions, such that in each one a node appears for the first or second time. Therefore, if a path $\bar{p}$ is constructed by the algorithm, then it can have at most $2n(n+1)$ nodes. Using n distinct nodes, we can construct at most $(n+1)^{2n(n+1)}$ paths with $2n(n+1)$ or fewer nodes. ◀

▶ **Lemma 6.** *K-Roots have the following two properties.*

1. *All the K-roots are discovered before the algorithm terminates. Moreover, they are discovered in the increasing order of their best heights.*
2. *Suppose that r is a K-root with a best height b. If $p[v,k]$ is a path (from any node v to any keyword k) that is exposed before the iteration that discovers r as a K-root, then $w(p[v,k]) \leq b$.*

Proof. We first prove Part 1. Suppose that a keyword k is reachable from node v. As long as $v.marks[k]$ is *active* at the beginning of the main loop (line 10), Lemma 4 implies that the queue Q contains (at least) one suffix of a minimal path from v to k. By Lemma 5, the algorithm constructs a finite number of paths. By Proposition 1, the same path can be inserted into the queue at most twice. Since the algorithm does not terminate while Q is not empty, $v.marks[k]$ must be changed to *visited* after a finite time. It thus follows that each K-root is discovered after a finite time.

Next, we show that the K-roots are discovered in the increasing order of their best heights. Let r_1 and r_2 be two K-roots with best heights b_1 and b_2, respectively, such that $b_1 < b_2$. Lemma 3 implies the following for r_i $(i = 1, 2)$. For all keywords $k \in K$, a minimal path from r_i to k changes $r_i.marks[k]$ from *active* to *visited*; that is, r_i is discovered as a K-root by minimal paths. Suppose, by way of contradiction, that r_2 is discovered first. Hence, a path with weight b_2 is removed from Q while Lemma 4 implies that a suffix with a weight of at most b_1 is still on Q. This contradiction completes the proof of Part 1.

Now, we prove Part 2. As shown in the proof of Part 1, a K-root is discovered by minimal paths. Let r be a K-root with best height b. Suppose, by way of contradiction, that a path $p[v, k]$, such that $w(p[v, k]) > b$, is exposed before the iteration, say i, that discovers r as a K-root. By Lemma 4, at the beginning of iteration i, the queue Q contains a suffix with weight of at most b. Hence, $p[v, k]$ cannot be removed from Q at the beginning of iteration i. This contradiction proves Part 2. ◀

▶ **Lemma 7.** *Suppose that node v is discovered as a K-root at iteration i. Let $p_1[v', k']$ and $p_2[v, k]$ be paths that are exposed in iterations j_1 and j_2, respectively. If $i < j_1 < j_2$, then $w(p_1[v', k']) \leq w(p_2[v, k])$. Note that k and k' are not necessarily the same and similarly for v and v'; moreover, v' has not necessarily been discovered as a K-root.*

Proof. Suppose the lemma is false. In particular, consider an iteration j_1 of the main loop (line 10) that violates the lemma. That is, the following hold in iteration j_1.

- Node v has already been discovered as a K-root in an earlier iteration (so, there are no frozen paths at v).
- A path $p_1[v', k']$ is exposed in iteration j_1.
- A path $p_2[v, k]$ having a strictly lower weight than $p_1[v', k']$ (i.e., $w(p_2[v, k]) < w(p_1[v', k'])$) will be exposed after iteration j_1. Hence, a proper suffix of this path is frozen at some node z during iteration j_1.

For a given v and $p_1[v', k']$, there could be several paths $p_2[v, k]$ that satisfy the third condition above. We choose one, such that its suffix is frozen at a node z that is closest from v. Since v has already been discovered as a K-root, z is different from v.

Clearly, $z.marks[k]$ is changed to *visited* before iteration j_1. By Lemma 3, a minimal path $p_m[z, k]$ does that. Let $p_s[v, z]$ be a minimal path from v to z.

Consider the path

$$\bar{p}[v, k] = p_s[v, z] \circ p_m[z, k].$$

Since both $p_s[v, z]$ and $p_m[z, k]$ are minimal, the weight of their concatenation (i.e., $\bar{p}[v, k]$) is no more than that of $p_2[v, k]$ (which is also a path that passes through node z). Hence, $w(\bar{p}[v, k]) < w(p_1[v', k'])$.

We now show that the conditions of Lemma 2 are satisfied at the beginning of iteration j_1 (i.e., j_1 corresponds to i in Lemma 2). Conditions 1–2 clearly hold. Condition 3 is satisfied because a suffix of $p_2[v, k]$ is frozen at z. Condition 4 holds, because of the choice of z and the

fact $w(\bar{p}[v,k]) < w(p_1[v',k'])$ that was shown earlier. Condition 5 holds, because otherwise $\bar{p}[v,k]$ would be unfrozen and $z.marks[k]$ would be *in-answer* rather than *visited*.

By Lemma 2, a suffix of $\bar{p}[v,k]$ is on the queue at the beginning of iteration j_1. This contradicts the assumption that the path $p_1[v',k']$ is removed from the queue at the beginning of iteration j_1, because $\bar{p}[v,k]$ (and, hence, any of its suffixes) has a strictly lower weight. ◀

▶ **Lemma 8.** *For all nodes $v \in V$, such that v is a K-root, the following holds. If z is a node on a simple path from v to some $k \in K$, then $z.marks[k] \neq$ visited when the algorithm terminates.*

Proof. The algorithm terminates when the test of line 10 shows that Q is empty. Suppose that the lemma is not true. Consider some specific K-root v and keyword k for which the lemma does not hold. Among all the nodes z that violate the lemma with respect to v and k, let z be a closest one from v. Observe that z cannot be v, because of the following two reasons. First, by Lemma 6, node v is discovered as a K-root before termination. Second, when a K-root is discovered (in lines 27–28), all its marks become *in-answer* in lines 29–31.

Suppose that $p_m[z,k]$ is the path that changes $z.marks[k]$ to *visited*. Let $p_s[v,z]$ be a minimal path from v to z. Note that $p_s[v,z]$ exists, because z is on a simple path from v to k. Consider the path

$$\bar{p}[v,k] = p_s[v,z] \circ p_m[z,k].$$

Suppose that the test of line 10 is **false** (and, hence, the algorithm terminates) on iteration i. We now show that the conditions of Lemma 2 are satisfied at the beginning of that iteration. Conditions 1–2 of Lemma 2 clearly hold. Conditions 3–4 are satisfied because of how z is chosen. Condition 5 holds, because otherwise $z.marks[k]$ should have been changed to *in-answer*.

By Lemma 2, a suffix of $\bar{p}[v,k]$ is on Q when iteration i begins, contradicting our assumption that Q is empty. ◀

▶ **Theorem 9.** *GTF is correct. In particular, it finds all and only answers to the query K by increasing height within $2(n+1)^{2n(n+1)}$ iterations of the main loop (line 10), where $n = |V|$.*

Proof. By Lemma 5, the algorithm constructs at most $(n+1)^{2n(n+1)}$ paths. By Proposition 1, a path can be inserted into the queue Q at most twice. Thus, the algorithm terminates after at most $2(n+1)^{2n(n+1)}$ iterations of the main loop.

By Part 1 of Lemma 6, all the K-roots are discovered. By Lemma 8, no suffix of a simple path from a K-root to a keyword can be frozen upon termination. Clearly, no such suffix can be on Q when the algorithms terminates. Hence, the algorithm constructs all the simple paths from each K-root to every keyword. It thus follows that the algorithm finds all the answers to K. Clearly, the algorithm generates only valid answers to K.

Next, we prove that the answers are produced in the order of increasing height. So, consider answers a_1 and a_2 that are produced in iterations j_1' and j_2, respectively. For the answer a_i $(i = 1, 2)$, let r_i and h_i be its K-root and height, respectively. In addition, let b_i be the best height of r_i $(i = 1, 2)$.

Suppose that $j_1' < j_2$. We have to prove that $h_1 \leq h_2$. By way of contradiction, we assume that $h_1 > h_2$. By the definition of best height, $h_2 \geq b_2$. Hence, $h_1 > b_2$.

Let $p_2[r_2,k]$ be the path of a_2 that is exposed (i.e., removed from Q) in iterations j_2. Suppose that $p_1[r_1,k']$ is a path of a_1, such that $w(p_1[r_1,k']) = h_1$ and $p_1[r_1,k']$ is exposed in the iteration j_1 that is as close to iteration j_1' as possible (among all the paths of a_1 from r_1 to a keyword with a weight equal to h_1). Clearly, $j_1 \leq j_1'$ and hence $j_1 < j_2$.

We now show that $w(p_1[r_1, k']) < h_1$, in contradiction to $w(p_1[r_1, k']) = h_1$. Hence, the claim that $h_1 \leq h_2$ follows. Let i be the iteration that discovers r_2 as a K-root. There are two cases to consider as follows.

Case 1: $i < j_1$. In this case, $i < j_1 < j_2$, since $j_1 < j_2$. By Lemma 7, $w(p_1[r_1, k']) \leq w(p_2[r_2, k])$. (Note that we apply Lemma 7 after replacing v and v' with r_2 and r_1, respectively.) Hence, $w(p_1[r_1, k']) < h_1$, because $w(p_2[r_2, k]) \leq h_2$ follows from the definition of height and we have assumed that $h_1 > h_2$.

Case 2: $j_1 \leq i$. By Part 2 of Lemma 6, $w(p_1[r_1, k']) \leq b_2$. Hence, $w(p_1[r_1, k']) < h_1$, because we have shown earlier that $h_1 > b_2$.

Thus, we have derived a contradiction and, hence, it follows that answers are produced by increasing height. ◀

▶ **Corollary 10.** *The running time of the algorithm GTF is* $O\left(kn(n+1)^{2kn(n+1)+1}\right)$*, where* n *and* k *are the number of nodes in the graph and keywords in the query, respectively.*

Proof. The most expensive operation is a call to **produceAnswers**$(v.paths, p)$. By Lemma 5, there are at most $(n+1)^{2n(n+1)}$ paths. A call to the procedure **produceAnswers**$(v.paths, p)$ considers all combinations of $k-1$ paths plus p. For each combination, all its k paths are traversed in linear time. Thus, the total cost of one call to **produceAnswers**$(v.paths, p)$ is $O\left(kn(n+1)(n+1)^{(k-1)2n(n+1)}\right)$. By Theorem 9, there are at most $2(n+1)^{2n(n+1)}$ iterations. Hence, the running time is $O\left(kn(n+1)^{2kn(n+1)+1}\right)$. ◀

5 Summary of the Experiments

In this section, we summarize our experiments. The full description of the methodology and results is given in Appendix B of [5]. We performed extensive experiments to measure the efficiency of GTF. The experiments were done on the Mondial[3] and DBLP[4] datasets.

To test the effect of freezing, we ran the naive approach (described in Section 3.1) and GTF on both datasets. We measured the running times of both algorithms for generating the top-k answers ($k = 100, 300, 1000$). We discovered that the freezing technique gives an improvement of up to about one order of magnitude. It has a greater effect on Mondial than on DBLP, because the former is highly cyclic and, therefore, has more paths (on average) between a pair of nodes. Freezing has a greater effect on long queries than short ones. This is good, because the bigger the query, the longer it takes to produce its answers. This phenomenon is due to the fact that the average height of answers increases with the number of keywords. Hence, the naive approach has to construct longer (and probably more) paths that do not contribute to answers, whereas GTF avoids most of that work.

In addition, we compared the running times of GTF with those of BANKS [1, 7], BLINKS [6], SPARK [9] and ParLMT [4]. The last one is a parallel implementation of [3]; we used its variant ES (early freezing with single popping) with 8 threads. BANKS has two versions, namely, MI-BkS [1] and BiS [7]. The latter is faster than the former by up to one order of magnitude and we used it for the running-time comparison.

GTF is almost always the best, except in two particular cases. First, when generating $1,000$ answers over Mondial, SPARK is better than GTF by a tiny margin on queries with 9 keywords, but is slower by a factor of two when averaging over all queries. On DBLP, however,

[3] http://www.dbis.informatik.uni-goettingen.de/Mondial/
[4] http://dblp.uni-trier.de/xml/

SPARK is slower than GTF by up to two orders of magnitude. Second, when generating 100 answers over DBLP, BiS is slightly better than GTF on queries with 9 keywords, but is 3.5 times slower when averaging over all queries. On Mondial, however, BiS is slower than GTF by up to one order of magnitude. All in all, BiS is the second best algorithm in most of the cases. The other systems are slower than GTF by one to two orders of magnitude.

Not only is our system faster, it is also increasingly more efficient as either the number of generated answers or the size of the data graph grows. This may seem counterintuitive, because our algorithm is capable of generating all paths (between a node and a keyword) rather than just the minimal one(s). However, our algorithm generates non-minimal paths only when they can potentially contribute to an answer, so it does not waste time on doing useless work. Moreover, if only minimal paths are constructed, then longer ones may be needed in order to produce the same number of answers, thereby causing more work compared with an algorithm that is capable of generating all paths.

GTF does not miss answers (i.e., it is capable of generating all of them). Among the other systems we tested, ParLMT [4] has this property and is theoretically superior to GTF, because it enumerates answers with polynomial delay (in a 2-approximate order of increasing height), whereas the delay of GTF could be exponential. In our experiments, however, ParLMT was slower by two orders of magnitude, even though it is a parallel algorithm (that employed eight cores in our tests). Moreover, on a large dataset, ParLMT ran out of memory when the query had seven keywords. The big practical advantage of GTF over ParLMT is explained as follows. The former constructs paths incrementally whereas the latter (which is based on the Lawler-Murty procedure [8, 10]) has to solve a new optimization problem for each produced answer, which is costly in terms of both time and space.

A critical question is how important it is to have an algorithm that is capable of producing all the answers. We compared our algorithm with BANKS. Its two versions only generate answers consisting of minimal paths and, moreover, those produced by BiS have distinct roots. BiS (which is overall the second most efficient system in our experiments) misses between 81% (on DBLP) to 95% (on Mondial) of the answers among the top-100 generated by GTF. MI-BkS misses much fewer answers, that is, between 1.8% (on DBLP) and 32% (on Mondial), but it is slower than BiS by up to one order of magnitude. For both versions the percentage of misses increases as the number of generated answers grows. This is a valid and significant comparison, because our algorithm generates answers in the same order as BiS and MI-BkS, namely, by increasing height.

6 Conclusions

We presented the GTF algorithm for enumerating, by increasing height, answers to keyword search over data graphs. Our main contribution is the freezing technique for avoiding the construction of (most if not all) non-minimal paths until it is determined that they can reach K-roots (i.e., potentially be parts of answers). Freezing is an intuitive idea, but its incorporation in the GTF algorithm involves subtle details and requires an intricate proof of correctness. In particular, cyclic paths must be constructed, although they are not part of any answer. For efficiency's sake, however, it is essential to limit the creation of cyclic paths as much as possible, which is accomplished by lines 24 and 36 of Figure 6.

Freezing is not merely of theoretical importance. Our extensive experiments (described in Section 5 and Appendix B of [5]) show that freezing increases efficiency by up to about one order of magnitude compared with the naive approach (of Section 3.1) that does not use it.

The experiments of Section 5 and Appendix B of [5] also show that in comparison to other systems, GTF is almost always the best, sometimes by several orders of magnitude.

Moreover, our algorithm is more scalable than other systems. The efficiency of GTF is a significant achievement especially in light of the fact that it is complete (i.e., does not miss answers). Our experiments show that some of the other systems sacrifice completeness for the sake of efficiency. Practically, it means that they generate longer paths resulting in answers that are likely to be less relevant than the missed ones.

The superiority of GTF over ParLMT is an indication that polynomial delay might not be a good yard stick for measuring the practical efficiency of an enumeration algorithm. An important topic for future work is to develop theoretical tools that are more appropriate for predicting the practical efficiency of those algorithms.

References

1. Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.
2. Joel Coffman and Alfred C. Weaver. An empirical performance evaluation of relational keyword search techniques. *IEEE Trans. Knowl. Data Eng.*, 26(1):30–42, 2014.
3. Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, 2008.
4. Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Optimizing and parallelizing ranked enumeration. *PVLDB*, 2011.
5. Konstantin Golenberg and Yehoshua Sagiv. A practically efficient algorithm for generating answers to keyword search over data graphs. *arXiv*, 2015. URL: `http://arxiv.org/abs/1512.06635`.
6. Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, 2007.
7. Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.
8. E. L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 1972.
9. Yi Luo, Wei Wang, Xuemin Lin, Xiaofang Zhou, Jianmin Wang, and Keqiu Li. SPARK2: Top-k keyword query in relational databases. *IEEE Trans. Knowl. Data Eng.*, 2011.
10. K. G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 1968.
11. Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE*, 2009.